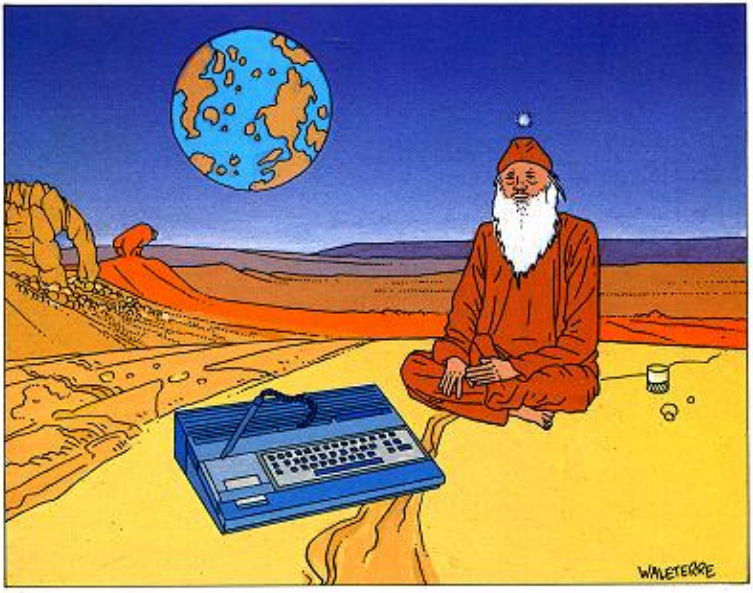
TO7 TO 7

LA FACE CACHEE DU TO-7/TO7-70

Jean-Baptiste Touchard



cedic/nathan

La face cachée du TO7

Jean Baptiste Touchard

cedic/nathan

Couverture et illustrations : Walter Lalonde
Maquette : Alain Dufourcq et Véronique Chabert d'Hieres
Montage : Michèle Beaucamp

Ce volume porte la référence
ISBN 2-7124-0563-3

L'Editeur remercie **Thomson Semi-conducteurs** pour l'autorisation qui lui a été accordée de reproduire des extraits de son *Catalogue Microprocesseurs et Périphériques* aux pages 143 à 159 de cet ouvrage.

Toute reproduction, même partielle, de cet ouvrage est interdite. Une copie ou reproduction par quelque procédé que ce soit, photographie, photocopie, microfilm, bande magnétique, disque ou autre, constitue une contrefaçon passible des peines prévues par la loi du 11 mars 1957 sur la protection des droits d'auteur.

© CEDIC 1984

CEDIC, 32, boulevard Saint-Germain, 75005 - PARIS

Sommaire

Avertissements 7

La mémoire du TO7 9

Plan général 10
Mémo 7 11
La mémoire d'écran 11
RAM 12
Rien 12
Moniteur 12
E/S 13

PEEK et POKE 15

PEEK (M) 16
POKE M, D 16

Constantes 18

Base 2, 10 et 16 19
La numération de position en trois exemples 19
Opérations 20
Conversions 22
Et les décimales ? 26
Le binaire signé 27

Variables 31

Variables entières 32
Nombres en simple précision 33
Double précision 35
Rangement en mémoire 35
Les variables chaînes de caractères 39

Interlude 42

PEEK et POKE en comprenant ! 43

PEEK mémoire 44
Où est le programme BASIC ? 50
Les mots du BASIC 52
Trucages divers 56
Interlude 63

Codes ASCII et normes vidéotex 64

La routine PUTCH 65
Les caractères de contrôle (CO) 66
L'escapage 68
La commande US 69
Les trois grilles GØ, G1, G2 71
Un programme d'illustration 73
Caractères définis par l'utilisateur 74
Loupes et écritures penchées 75

Curseur et page graphique 79

La mémoire d'écran 80
Interlude 82
Les couleurs 83
Le curseur 88
Bidouilles graphiques 89
Interlude 95

Bruits 96

Bzzs, couacs et pouêts 97
Bip, bip 100
Truc 1 104
Truc 2 104
Truc 3 105
Un autre son 105

Pia, pia, pia ! 107

L'interface des jeux 108
En entrée 109
En sortie 109
Musique (bis) 111

Greffes, USR 114

Enquête dans la ROM BASIC 120

A la recherche de constantes mathématiques 121
A l'aventure dans le Math Pack 122
Un coup d'œil vers 1ABF 124
Demain, je désassemble le haut ! 126
Chercher l'erreur 129
De mieux en mieux 131
Postlude 132

Annexes 133

Les routines moniteur 133
Les connecteurs 138
Les codes 6809 142

Avertissements

Premier avertissement

Ce livre est sans danger !

Vous y découvrirez beaucoup de « trucs » qui vous paraîtront parfois assez osés. Plus d'une fois, il vous arrivera de « planter » votre micro-ordinateur. Toutefois, vous ne risquez rien de définitif ! Dans le pire des cas, il vous suffit d'éteindre le courant, puis de rallumer. Peut-être aurez-vous perdu votre programme, mais si vous comprenez pourquoi votre machine s'est plantée, vous n'aurez pas perdu votre temps.

Deuxième avertissement

Ce livre est dangereux !

Il contient une matière assez originale dont la teneur s'apparente principalement au temps. Certains diront à la perte de temps. L'auteur a parfaitement conscience d'avoir passé de nombreuses heures à la recherche de rien ou plutôt de la simple compréhension de phénomènes complexes. Il est très perplexe quant à l'utilisation que l'on peut faire de ces connaissances. Les utiliser pour mieux programmer ? Peut-être, mais c'est avant tout le plaisir de comprendre comment ça marche qui l'a poussé à mener ces investigations. Ce livre ne prétend pas expliquer en détail le fonctionnement du TO7, il faudrait en effet sans doute quelques

dizaines de kilos de papier pour donner l'ensemble des schémas électriques, les listings des différents programmes assembleurs qui tournent dans le TO7 ainsi que les explications du fonctionnement de ces différents éléments. Ce livre offre plutôt des pistes et des points de départ pour des investigations, des trucs, des gags.

L'auteur manipule principalement le BASIC pour développer ses exemples. A l'occasion, il utilisera un peu d'assembleur avec l'idée que les mini-programmes assembleurs présentés peuvent fort bien servir de première approche de ce langage. Il s'agit toutefois d'une approche non structurée; on ne donnera pas dans ce livre un cours d'assembleur, mais plutôt, là encore, de petits éléments de réflexion.

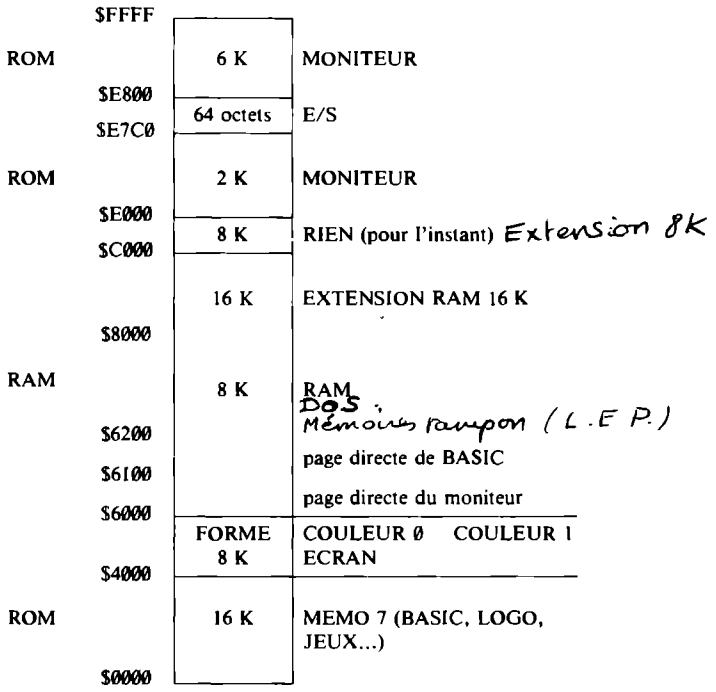
La mémoire du TO7

Plan général

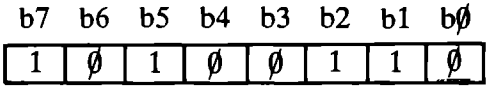
Le 6809 peut adresser 65 536 octets. On dit que l'espace adressable est de 64 Ko.

Chaque octet de mémoire porte un numéro : son adresse. Pour des tas de raisons, il est pratique de s'exprimer en hexadécimal quand on donne l'adresse d'un octet ou quand on parle de la valeur qu'il contient (voir le chapitre suivant).

Tableau d'adresses



Repères $1ØØ : 256
$1ØØØ : 4Ø96 (4 K)
$4ØØ :1Ø24 (1 K)



Un octet

Mémo 7

La zone mémoire $ØØØØ - $3FFF contient la ROM [1] ou mémoire à lecture seule, enfichée dans le Mémo 7. Les premières adresses ($Ø - $2Ø) contiennent le titre de la ROM tel qu'il est affiché au menu lors de l'initialisation.

On trouve ensuite aux adresses $ØØ1C - $ØØ1D sur deux octets l'adresse du point d'entrée « à chaud » de la cartouche et en $ØØ1E - $ØØ1F, de la même manière, l'adresse du point d'entrée « à froid » de la cartouche. Vient ensuite tout ce qui fait l'originalité de la cartouche, les mots connus et les divers programmes qui permettent de mener à bien les actions provoquées par ces mots.

Le Mémo 7 peut occuper 16 K; il peut naturellement occuper moins de 16 K et c'est en général le cas pour les jeux. Il peut éventuellement occuper plus de 16 K en utilisant la technique du « bank switching » qui permet d'avoir plusieurs données aux mêmes adresses, plusieurs banques de mémoires commutables. C'est le cas pour la cartouche LSE qui ne pourrait pas tenir sur 16 K.

Les informations sur le contenu des Mémo 7 sont très rares. Nous chercherons tout de même à voir comment on peut essayer de s'y retrouver au chapitre « Enquête dans la ROM BASIC ».

La mémoire d'écran

La zone mémoire $4ØØØ - $5FFF correspond à l'écran. Cette zone est constituée de 8 192 octets de RAM. Les 8 000 premiers correspondent effectivement à ce qui est affiché à l'écran, les 192 suivants ne servent apparemment à rien... pour l'instant. La structure de la mémoire d'écran est assez particulière et nous y consacrerons un petit moment. Notons pourtant dès à présent qu'il y a trois plans mémoires aux mêmes adresses sur le TO7 et deux sur le TO7/70 et le MO5 qui contiennent respectivement les formes et les couleurs de ce qui est affiché à l'écran. Ces trois plans peuvent être (et sont !) travaillés indépendamment par « bank switching ».

(Changement de bloc de mémoire)

1.ROM : « Read Only Memory » ou mémoire à lecture seule. Les informations contenues dans ces mémoires sont gravées de manière inaltérable.

RAM

La zone $6ØØØ - $7FFF, étendue jusqu'à $BFFF si l'on dispose d'une extension mémoire, est la zone RAM [1] utilisateur. Cette zone de 8 K ou de 24 K est en général organisée par le langage sous le contrôle duquel on est. Le moniteur se réserve toujours les 256 premiers octets ($6ØØØ - $6ØFF). Le BASIC, par exemple, se réservera un peu de place pour sa « page directe » ($61ØØ - $61FF), puis pour ses mémoires tampons avec l'extérieur (lecteur-enregistreur de programmes), puis éventuellement plus de 8 K pour y placer le DOS (« disk operating system ») avec tous ses ordres : SAVE, LOAD, DIR, FIELD, SWAP, FILL, etc.

Viennent enfin l'image du programme entré par l'utilisateur de la machine, les valeurs des variables et sûrement d'autres petites choses plus ou moins bien connues.

Rien

Sur TO7, il reste ici 8 K — la zone $CØØØ-$DFFF — qui ne correspondent à rien. Rien n'est branché au bout de ces fils ! Gageons pourtant qu'un bricoleur va nous servir rapidement une super-extension-mémoire qui viendra profiter de ces adresses pour l'instant inutilisées. Sur TO7-70, cette zone contient de la RAM. On peut d'ailleurs accéder à deux pages des 16 K situées entre les adresses AØØØ et DFFF par bank switching.

Moniteur

Cette zone de près de 8 K, $EØØØ - $E7BF et $E8ØØ - $FFFF, contient le moniteur. C'est là que résident les programmes qui pilotent les divers éléments qui font l'originalité de la machine. Ces programmes sont en général appelés par l'intermédiaire du programme situé dans le Mémo 7; pourtant, l'utilisateur peut s'en servir directement en assembleur.

Les routines offertes par le moniteur sont :

1.RAM : « Random Access Memory » ou mémoire à accès aléatoire; on peut y écrire et y lire. Cette mémoire est « vidée » lors d'une coupure de courant.

PUTCH GETSC INITSC CHPLOT	Gestion de l'écran alphanumérique
PLOTXY DRAWXY GETPT	Gestion de l'écran graphique
GETCH KEYST	Gestion du clavier
K7CONT	Gestion du LEP
RSCONT	Gestion de l'interface série-parallèle
NOTE	Générateur musical
LPINT GETLP	Gestion du crayon optique
DKCONT DKBOOT FORMAT	Gestion du lecteur de disques
JOYSTICK	Gestion des manettes de jeu

E/S

Cette zone, $E7CØ - $E7FF, située au milieu de la ROM moniteur, contient les adresses d'entrée-sortie. A ces adresses, on ne trouve pas à proprement parler de la mémoire mais plutôt des circuits connus sous le nom de PIA, ACIA, TIMER... Ces circuits fonctionnent souvent à la manière d'interrupteurs et servent au microprocesseur à communiquer avec l'extérieur (écran, clavier, buzzer...).

Ainsi, par exemple, le bit Ø de l'adresse $E7C3 permet de sélectionner le travail dans le plan mémoire formes ou dans le plan mémoire couleurs de l'écran graphique. Sur le TO7, le bit 3 de cette même adresse $E7C3 correspond à l'interrupteur majuscules/ minuscules du clavier. Ainsi,

```
POKE &HE7C3,PEEK(&HE7C3) OR  &B00001000
```

place la machine en minuscules, et

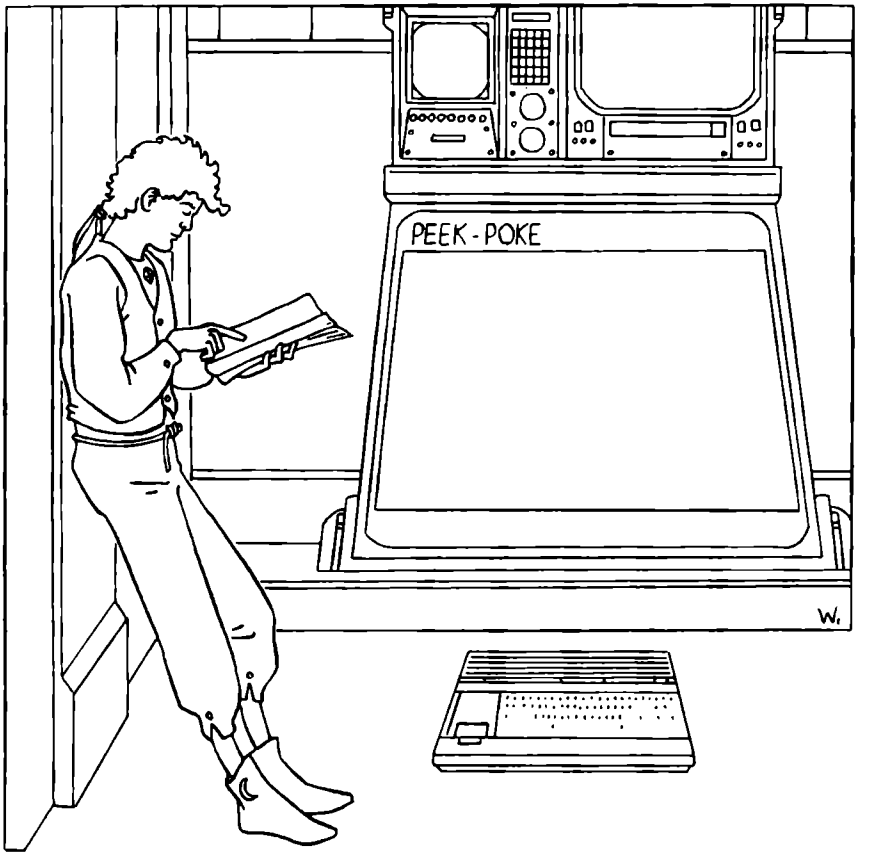
```
poke &he7c3,peek(&he7c3) and &b11110111
Ok
```

place la machine en majuscules.

Toujours sur le TO7, les bits 4, 5 et 6 de cette même adresse, $E7C3, déterminent la couleur du bord de l'écran, et le bit 2 permet de commuter les deux pages couleurs Ø et 1. On verra au chapitre suivant un mini-programme qui utilise cela pour rendre le TO7 fou, l'espace de 256 instants.

sur MO5 ⇒ $ A7CØ (42944)

PEEK et POKE



PEEK (M)

« To peek » veut dire « jeter un coup d'œil ». PEEK sert à étudier le contenu des mémoires. Les 65 536 octets de la mémoire peuvent contenir une information codée de Ø à 255. Cette information peut évidemment représenter une valeur numérique, mais elle peut également représenter autre chose : un caractère affichable (lettre, chiffre ou symbole), une instruction, une portion de dessin ou n'importe quoi, convenu à l'avance.

Le paramètre M peut prendre toutes valeurs entre Ø et 65 535. PRINT PEEK (Ø) donne le contenu de la mémoire Ø qui vaut 32 (c'est toujours ainsi car la mémoire Ø est en ROM). PRINT PEEK (16 384) peut valoir différentes choses car la mémoire 16 384 est en RAM : c'est la première mémoire de la zone écran. Après une remise à zéro (RAZ), PRINT PEEK (16 384) donne Ø.

POKE M,D

« To poke » veut dire « enfoncer ». POKE sert à déposer une valeur D dans une mémoire M. Cette valeur doit être naturellement comprise entre Ø et 255. Le numéro de la mémoire peut être n'importe lequel entre Ø et 65 535. Toutefois, POKE sera sans effet sur la ROM et risque de « planter le système » s'il est pratiqué sans précaution sur une mémoire RAM. Cependant, POKE ne causera jamais de dommages définitifs : au pire, il provoquera une panne logicielle majeure qui sera oubliée dès que l'on aura éteint puis rallumé l'ordinateur. Bien entendu, on risque tout de même de perdre les informations présentes en RAM à ce moment.

```
PRINT PEEK(0)
 32
OK
POKE 0,12          Ce POKE est sans effet
OK                 sur la ROM...
PRINT PEEK(0)
 32
OK
```

RAZ

ENTREE

ENTREE

ENTREE

```
PRINT PEEK(16384)
 0
OK
POKE 16384,255
OK
PRINT PEEK(16384)
 255
OK
POKE 16384,67
OK
PRINT PEEK(16384)
 67
OK
```

Ici, on « poke » de la RAM, les valeurs déposées sont bien là, comme le montre PEEK. De plus, on « voit » ces valeurs sur les huit premiers points en haut à gauche de l'écran, qui correspondent justement à cette mémoire.

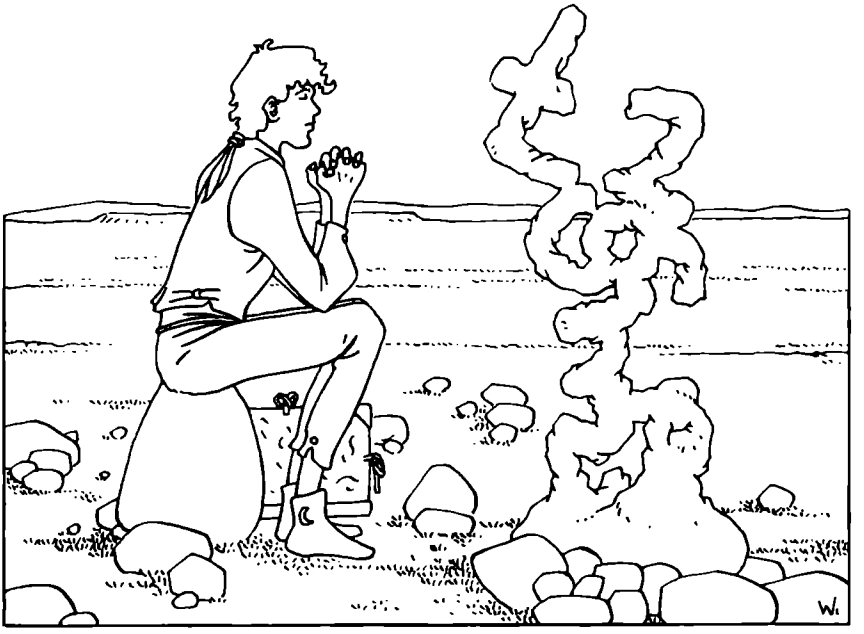
Il en ira de même avec les 8000 mémoires dont les adresses vont de 16 384 à 24 383. On peut essayer de « poker » &HE7C3 avec diverses valeurs, on verra alors peut-être le bord changer de couleur. L'écran pourra sembler pris d'une grande panique ainsi que le voyant « minuscule » du clavier.

Le programme ci-dessous vous garantit un beau résultat :

```
10 FOR I=0 TO 255
20 POKE &HE7C3,I
30 NEXT I
```

&HE7C3 représente 59 331 en hexadécimal.

Constantes



Bases 2, 1Ø, et 16

L'ordinateur manipule des données ! Ces données peuvent représenter de multiples choses mais en tout état de cause, elles sont rangées dans la mémoire sous forme de mots binaires. Les codages, qui permettent de donner à tel ou tel mot telle ou telle signification, sont ensuite affaire de conventions.

Système binaire
Base 2

Deux chiffres : Ø et 1
Notation BASIC : &BIØIØIIØI
Notation assembleur : %IØIØIIØI

Système hexadécimal
Base 16

16 chiffres : Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F,
Notation BASIC : &H6F3F
Notation assembleur : $6F3F
et parfois : 6F3FH
et quand il n'y a pas de doute, on écrit de temps en temps 6F3F tout simplement.

Système décimal
Base 10

Dix chiffres : Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9
La notation des nombres exprimés en base 10 est naturelle : 1 515, 732...

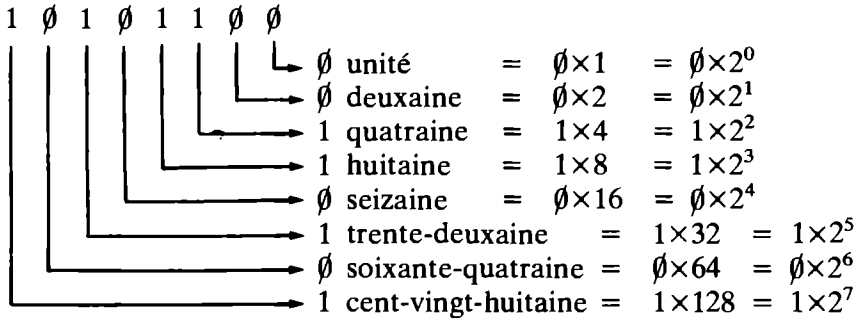
La numération de position en trois exemples

Un nombre décimal (base 1Ø) :

1 9 8 3

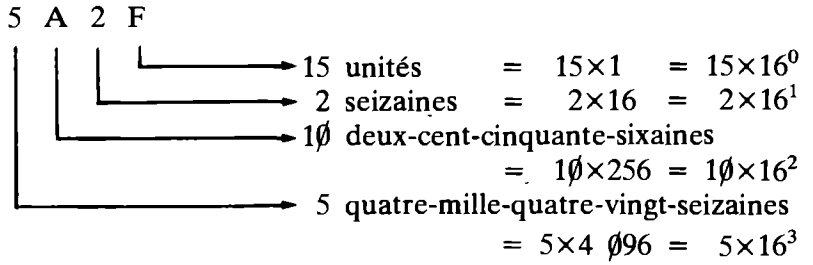
3 unités	=	3 × 1	=	3 × $1Ø^0$
8 dizaines	=	8 × 1Ø	=	8 × $1Ø^1$
9 centaines	=	9 × 1ØØ	=	9 × $1Ø^2$
1 millier	=	1 × 1ØØØ	=	1 × $1Ø^3$

Un nombre binaire (base 2) :



&B1Ø1Ø11ØØ = 172

Un nombre hexadécimal (base 16) :



&H5A2F = 23 087

Les opérations

Tables d'opérations binaires

Addition

+	Ø	1
Ø	Ø	1
1	1	1Ø

Multiplication

×	Ø	1
Ø	Ø	Ø
1	Ø	1

C'est tout !

DEC	BIN	HEXA
13 + 5 18	1101 + 101 10010	$800 + $800 $1000

13 − 2 11	1101 − 10 1011
DEC	BIN

13 × 6 78	1101 × 110 0000 1101 1101 1001110
DEC	BIN

328 | 11 108 | 29 9	10100.1000 | 1011 10011 | 11101 10000 10100 1001
DEC	BIN

Décaler un nombre d'un rang vers la gauche revient à :

- le multiplier par 10 en base 10,
  28 ⟶ 280

- le multiplier par 2 en base 2,
  [&B110 ⟶ &B1100
    6 ⟶ 12]

- le multiplier par 16 en base 16.
  [&H10 ⟶ &H100
    16 ⟶ 256]

Conversions

Décimal	Binaire	Hexadécimal	
ø	øøøø	ø	
2	øø1ø	2	
3	øø11	3	
4	ø1øø	4	
5	ø1ø1	5	
6	ø11ø	6	
7	ø111	7	
8	1øøø	8	
9	1øø1	9	
1ø	1ø1ø	A	
11	1ø11	B	
12	11øø	C	
13	11ø1	D	
14	111ø	E	
15	1111	F	
16	1øøøø	1ø	
255	1111 1111	FF	Un octet plein à craquer
2^8 256	*	1øø	
2^{10} 1ø24	*	8øø	1K
2^{14} 4ø96	*	1øøø	4K
2^{16}-1 65535	*	FFFF	Plus haute adresse disponible sur 16 bits
2^{16} 65536	*		
2^{20} 1048576	*		

* Notations rarement utilisées à cause de l'encombrement.

Deux et seize :

Pour convertir du binaire en hexadécimal et vice versa, c'est très simple. A chaque chiffre hexadécimal, on fait correspondre quatre chiffres binaires selon la table ci-dessus :

1ø1ø	11øø
A	C

&B1ø1ø = &HAC

&H5A2F = &B ø1ø1 1ø1ø øø1ø 1111

Dix et les autres

Pour convertir de la base 2 en base 10, on utilise la grille des puissances de 2 :

128	64	32	16	8	4	2	1
1	Ø	1	Ø	1	1	Ø	Ø

```
  128
+  32
+   8
+   4
-----
  172
```

&B1Ø1Ø11ØØ = 172

De même, pour convertir de la base 16 en base 10, on utilise la grille des puissances de 16 :

4Ø96	256	16	1
5	A	2	F

```
   15
+  2×16
+1Ø×256
+ 5×4 Ø96
---------
```

```
   15
   32
 2 56Ø
2Ø 48Ø
---------
```

&H5A2F = 23 087 23 Ø87

Mais heureusement l'ordinateur connaît les nombres hexadécimaux.

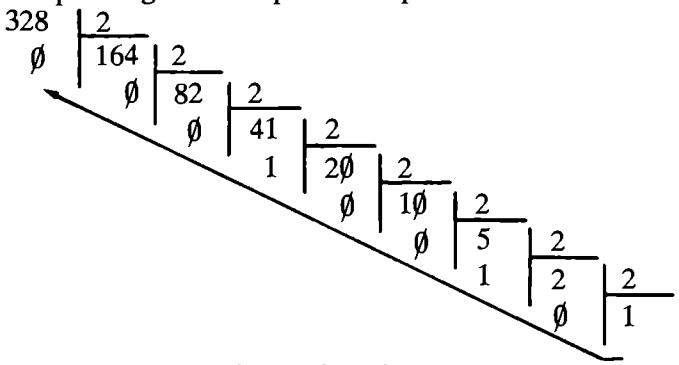
Pour convertir de la base 10 en base 2, on peut utiliser la grille par soustraction :

328
Il y va 256, reste 72
Il y va 64, reste 8

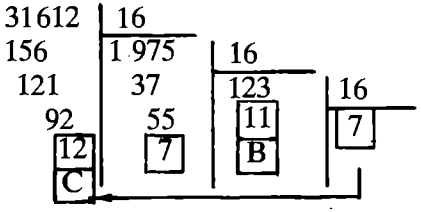
512	256	128	64	32	16	8	4	2	1
Ø	1	Ø	1	Ø	Ø	1	Ø	Ø	Ø

328 = &B1Ø1ØØ1ØØØ

On peut également procéder par division :



Pour passer de la base 10 en base 16, c'est tout aussi compliqué :



31 612 = &H7B7C

Mais heureusement, l'ordinateur connaît la fonction HEX$ qui fait ce travail.

```
PRINT HEX$(41672)
A2C8

PRINT &HA2C8
 41672

PRINT VAL("&H"+"A2C8")
 41672

PRINT &B1010001011001000
 41672
```

En pratique, on utilisera VAL et HEX$ pour passer du décimal à l'hexadécimal, les conversions en binaire étant rarement utiles et quasi immédiates avec l'hexadécimal.

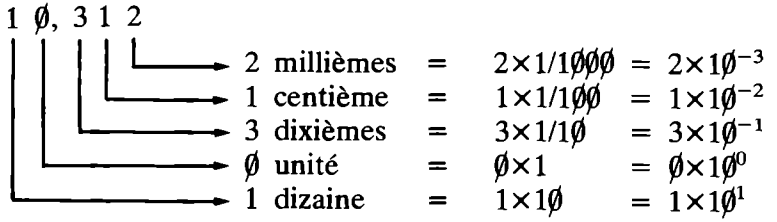
```
10 INPUT I
15 B$=""
20 Q=I\2
25 R$="0"
30 IF (I MOD 2)=1 THEN R$="1"
35 B$=R$+B$
40 I=Q
45 IF I<>0 THEN 20
50 B$="&B"+B$
55 PRINT B$
```

```
 0
&B0

 1
&B1

 2
&B10

 3
&B11

 4
&B100

 8
&B1000

 9
&B1001

 10
&B1010

 11
&B1011

 12
&B1100

 13
&B1101

 14
&B1110

 15
&B1111

 16
&B10000

 255
&B11111111

 256
&B100000000

 1515
&B10111101011
```

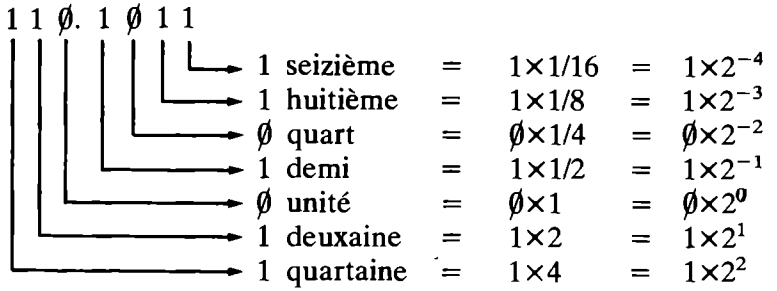
```
0   &H0   &B0
1   &H1   &B1
2   &H2   &B10
3   &H3   &B11
4   &H4   &B100
5   &H5   &B101
6   &H6   &B110
7   &H7   &B111
8   &H8   &B1000
9   &H9   &B1001
10  &HA   &B1010
11  &HB   &B1011
12  &HC   &B1100
13  &HD   &B1101
14  &HE   &B1110
15  &HF   &B1111
16  &H10  &B10000
17  &H11  &B10001
18  &H12  &B10010
19  &H13  &B10011
20  &H14  &B10100
21  &H15  &B10101
22  &H16  &B10110
23  &H17  &B10111
```

Et les décimales... ?

En base 10



En base 2, c'est pareil :



&B11Ø.1Ø11 = 6,6875

Note : si les nombres à virgules écrits en base 2 donnent toujours une suite décimale finie en base 10, il n'en va pas de même dans l'autre sens :

1/10 = 1/16 + 1/32 + 1/256 + 1/512 +....
Ø.1 = &BØ.ØØØ11ØØ11ØØ11.....[1]

C'est ce qui explique (en partie) certains résultats étranges ou surprenants du style de ceux que donne un programme comme celui-ci :

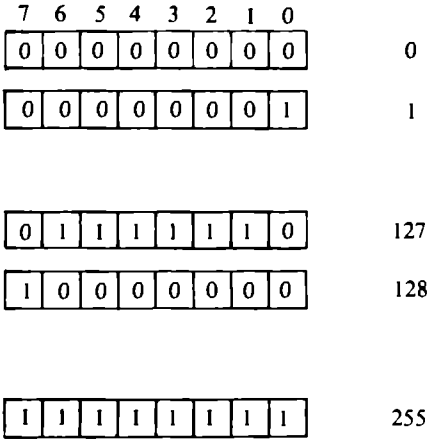
```
10 FOR I=0 TO 1 STEP 1/100
20 PRINT I
30 NEXT I
```

1. Attention : le BASIC ne connaît que les constantes binaires entières; la notation &BØ.Ø1Ø1 n'est pas avalée par BASIC.

Le binaire signé

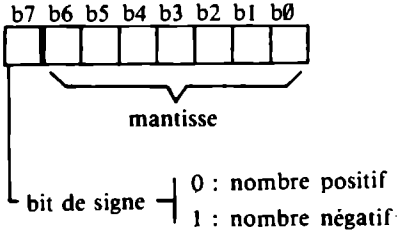
Sur un octet, il est possible de conserver 256 valeurs différentes correspondant à tous les arrangements possibles de Ø et de 1 dans les huit cases. Naturellement la première idée qui vient à l'esprit consiste à attribuer à ces 256 valeurs les valeurs numériques allant de Ø à 255 qui correspondent à la traduction décimale de la configuration binaire que l'on trouve dans l'octet.

Numéro de bit :

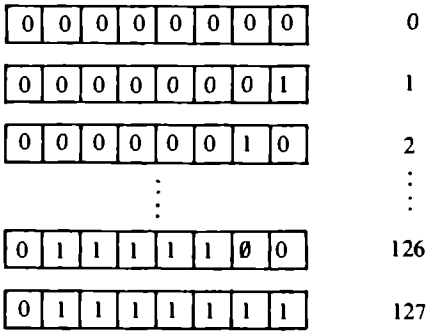


Et c'est d'ailleurs ainsi que travaillent les instructions PEEK et POKE.

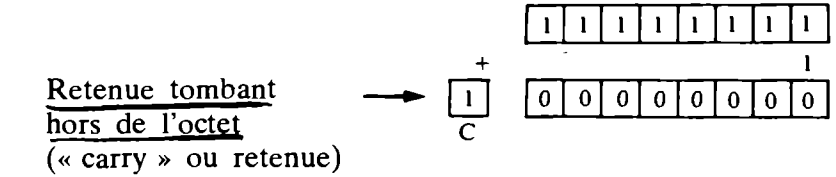
Certains informaticiens (géniaux !) eurent l'idée de ranger dans un octet les nombres de -128 à +127 (ce qui fait toujours 256 valeurs) en utilisant la notation « signée ». Le bit 7, connu également sous le nom « bit de poids fort » est alors le bit de signe, les bits suivants formant la « mantisse » du nombre :



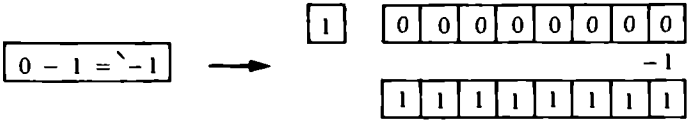
Les nombres positifs (de Ø à +127) sont notés de manière habituelle :



Les nombres négatifs sont notés « en complément à 2 ». Cette notation est faite pour que la somme de deux nombres opposés donne Ø ! Pour atteindre ce résultat, il faut d'abord remarquer que 255 + 1 = Ø. En effet, sur un octet, le jeu des retenues dans l'addition explique ce résultat :



On obtient donc l'opposé d'un nombre grâce à la soustraction sur huit bits (en imaginant un neuvième bit à 1 hors de l'octet), -1 s'écrira donc :



- 1 = 8HFF

(Ceci permet de comprendre pourquoi le vrai a pour valeur numérique -1; ØØØØØØØØ est le faux, 11111111 = − 1, son complémentaire logique, est le vrai). Techniquement, pour former l'opposé d'un nombre positif, on complémente tous les bits du nombre, puis on ajoute 1 :

0	1	1	0	1	0	1	0	106
1	0	0	1	0	1	0	1	$\overline{106}$
1	0	0	1	0	1	1	0	– 106

Quelques négatifs :

1	1	1	1	1	1	1	1	– 1
1	1	1	1	1	1	1	0	– 2
1	1	1	1	1	1	0	1	– 3
1	1	1	1	1	1	0	0	– 4
1	1	1	1	1	0	1	1	– 5
1	1	1	1	1	0	1	0	– 6
1	1	1	1	1	0	0	1	– 7
1	1	1	1	1	0	0	0	– 8

1	0	0	0	0	0	0	0	– 128

Quelques opérations

+ 1Ø6			Ø	1	1	Ø	1	Ø	1	Ø
+ (-1Ø6)			1	Ø	Ø	1	Ø	1	1	Ø
Ø	$\boxed{1}$		Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Quand le microprocesseur travaille ainsi en complément à 2, il doit faire attention aux débordements (« overflow » : V) qui peuvent se produire quand un résultat dépasse 127 :

83	0	1	0	1	0	0	1	1	83
+ 72	0	1	0	0	1	0	0	0	+ 72
155	1	0	0	1	1	0	1	1	– 101

(Le fait que la différence entre le bon résultat (155) et le résultat trouvé (-101) soit de 256 n'est sans doute pas le fruit du hasard.) Le débordement sur le bit 7 sera diagnostiqué par l'indication V.

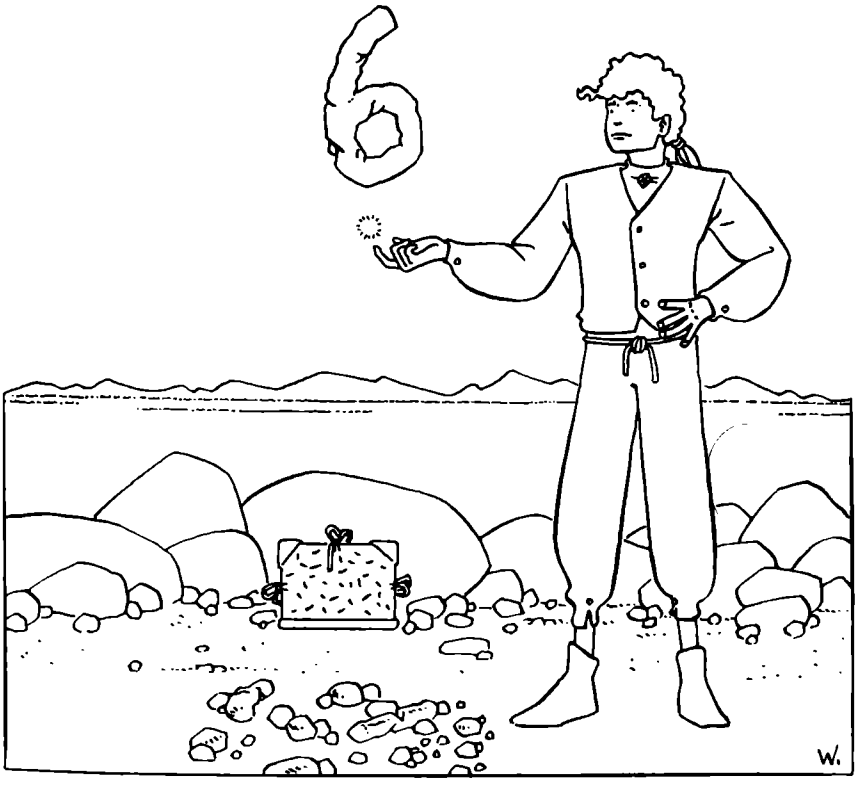
De − 16 à + 15 en binaire signé sur huit bits

0	0	0	0	1	1	1	1	15	1	1	1	1	1	1	1	1	− 1
0	0	0	0	1	1	1	0	14	1	1	1	1	1	1	1	0	− 2
0	0	0	0	1	1	0	1	13	1	1	1	1	1	1	0	1	− 3
0	0	0	0	1	1	0	0	12	1	1	1	1	1	1	0	0	− 4
0	0	0	0	1	0	1	1	11	1	1	1	1	1	0	1	1	− 5
0	0	0	0	1	0	1	0	10	1	1	1	1	1	0	1	0	− 6
0	0	0	0	1	0	0	1	9	1	1	1	1	1	0	0	1	− 7
0	0	0	0	1	0	0	0	8	1	1	1	1	1	0	0	0	− 8
0	0	0	0	0	1	1	1	7	1	1	1	1	0	1	1	1	− 9
0	0	0	0	0	1	1	0	6	1	1	1	1	0	1	1	0	− 10
0	0	0	0	0	1	0	1	5	1	1	1	1	0	1	0	1	− 11
0	0	0	0	0	1	0	0	4	1	1	1	1	0	1	0	0	− 12
0	0	0	0	0	0	1	1	3	1	1	1	1	0	0	1	1	− 13
0	0	0	0	0	0	1	0	2	1	1	1	1	0	0	1	0	− 14
0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	1	− 15
0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	− 16

Sur cette table, on constate que b7 = b6 = b5 = b4.

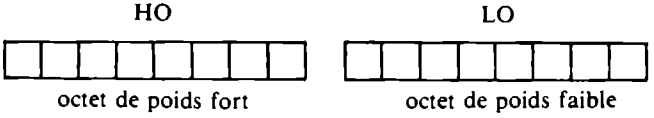
« Ce » bit représente le bit de signe.

Variables



Variables entières

Sur deux octets, c'est-à-dire sur 16 bits, il est possible de coder 65 536 (2^{16}) configurations binaires différentes. Si un ordinateur vous dit qu'il accepte de manipuler des entiers compris entre − 32 768 et + 32 767, vous devez comprendre à présent sous quelle forme ces nombres seront rangés : l'instruction VARPTR permet de pointer dans la mémoire le premier des octets dans lesquels est rangée la valeur numérique d'une variable. Le suffixe % après le nom d'une variable ou sa déclaration par une instruction du type DEFINT I impose à celle-ci d'être entière. La manipulation de variables entières permet, à l'occasion, d'économiser de la place et du temps de calcul.




```
10 INPUT "Valeur entiere :";A%
15 PTR=VARPTR(A%)
20 PRINT PEEK(PTR);PEEK(PTR+1)
25 PRINT
30 GOTO 10
```

pointer dans la mémoire = donner l'adresse

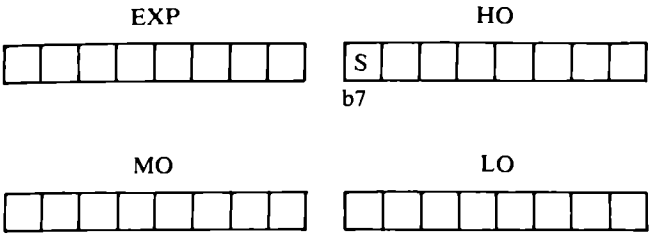
```
Valeur entière :? 0
 0  0

Valeur entière :? 1
 0  1

Valeur entière :? 345
 1  89

Valeur entière :? 255
 0  255

Valeur entière :? 256
 1  0

Valeur entière :? 30000
 117  48

Valeur entière :? -1
 255  255

Valeur entière :? -325
 254  187
```

On peut agréablement changer la ligne 2Ø pour :

```
20 PRINT HEX$(256*PEEK(PTR)+PEEK(PTR+1))
```

Nombres en simple précision

Le codage de la valeur numérique des variables manipulables par l'ordinateur en simple précision repose sur tout ce qui précède. La valeur d'une variable est codée sur quatre octets :



Le nombre est codé sur 32 bits en notation exponentielle en base 2. EXP contient l'exposant, HO, MO et LO contiennent les vingt-quatre bits de la mantisse, c'est-à-dire les chiffres situés à droite du point décimal.

Quelques conventions :
- Ø est représenté par un nombre pour lequel EXP = Ø. La mantisse est alors sans importance.
- L'exposant est décalé de $8Ø et noté en binaire signé. Ainsi :
EXP = $8Ø correspond à 2↑Ø
EXP = $81 correspond à 2↑1
EXP = $7F correspond à 2↑-1
- La mantisse a toujours la même structure : partie entière nulle, point décimal, un « 1 » juste à droite du point décimal, puis les chiffres suivant à droite. Il est donc possible de sous-entendre ce « 1 ». Le bit ainsi récupéré servira de bit de signe pour le nombre.

Le plus petit nombre positif non-nul connu par la machine sera donc :

EXP	HO	MO	LO	
1	Ø	Ø	Ø	(2.9874 E -39)

et le plus grand :

255	127	255	255	(1.7014 E +38)

Entre ces deux valeurs on a par exemple :

```
10 INPUT "Valeur numerique :";A
15 PTR=VARPTR(A)
20 PRINT PEEK(PTR);PEEK(PTR+1);PEEK(PTR+
2);PEEK(PTR+3)
25 PRINT
30 GOTO 10


Valeur numérique :? 0
 0  0  0  0
RUN
Valeur numérique :? 1
 129  0  0  0

Valeur numérique :? .5
 128  0  0  0

Valeur numérique :? .0625
 125  0  0  0

Valeur numérique :? 3
 130  64  0  0

Valeur numérique :? -3
 130  192  0  0

Valeur numérique :? 45.728
 134  54  233  120
```

La représentation de 45.728 peut se comprendre ainsi :

EXP	HO	MO	LO
134	54	233	12Ø
$86	$36	$E9	$78
	Ø0110110	11101001	Ø1111000

bit 7 de HO est à Ø : le nombre est positif.

Mantisse : Ø.1Ø11Ø11Ø1111Ø1ØØ1Ø1111ØØØ

$$Ø,7145 = ((128 + 54)*65\,536 + 233*256 + 120)/(2^{24})$$

↳ sous-entendu sous le bit de signe

Exposant : 134 − 128 = 6

Valeur : $2^6 * Ø,7145 = 45.728$

Double précision

Les nombres en double précision sont codés de manière analogue. La mantisse occupe sept octets au lieu de trois. Les noms habituels de ces octets sont :

EXP HO MO LO HOD MOD MLD LOD

Là où les variables « simple précision » comportaient environ six chiffres significatifs, les variables « double précision » en comportent seize.

En BASIC, pour que les calculs soient menés en double précision sur une variable, il faut ajouter le suffixe #.

LET P1# = 3.14159265358979...

Remarquez tout de même qu'il est très rare que l'on ait à utiliser ces variables coûteuses en place et en temps de calcul.

Rangement en mémoire

Pour en terminer avec les variables numériques, il peut être intéressant de savoir comment elles sont rangées en mémoire.

L'instruction BASIC VARPTR permet de connaître l'adresse à partir de laquelle on peut trouver les octets contenant la valeur d'une variable donnée. Il est tout de même intéressant de regarder un peu les octets qui précèdent afin de voir comment la variable est identifiée. En examinant la mémoire à coups de PEEK autour de l'adresse signalée par VARPTR, on peut voir des choses semblables à celles-ci :

```
10 AA=-1
11 BB%=2
12 CCC=45.726
13 DIM DDDD(3,2)
14 DV=256*PEEK(&H611E)+PEEK(&H611F)
15 FV=256*PEEK(&H6120)+PEEK(&H6121)
20 FOR M=DV-5 TO FV+20
30 PRINT HEX$(M),HEX$(PEEK(M))
40 NEXT M
OK
```

Voici les résultats obtenus et les explications :

DV-5 →	669A 20	
	669B 4D	
	669C 0	Fin du programme
	669D 0	
	669E 0	
DV →	669F 20	Type de variable : 2 (entière) : nom sur une lettre (Ø+1)
	66A0 41	A
	66A1 FF	} Valeur numérique = -1
	66A2 FF	
	66A3 21	Type de variable : 2 (entière); nom sur deux lettres (1+1)
	66A4 42	B
	66A5 42	B
	66A6 0	} Valeur numérique : 2
	66A7 2	
	66A8 42	Type de variable : 4 (flottant); nom sur trois lettres (2+1)
	66A9 43	C
	66AA 43	C
	66AB 43	C
	66AC 86	
	66AD 36	} Valeur numérique
	66AE E9	
	66AF 78	
	66B0 41	Type de variable : 4 (flottant); nom sur deux lettres (1+1)
	66B1 44	D
	66B2 56	V
	66B3 90	
	66B4 8	} Valeur numérique
	66B5 FB	
	66B6 0	
	66B7 41	Type de variable : 4 (flottant); nom sur deux lettres (1+1)
	66B8 46	F
	66B9 56	V
	66BA 90	
	66BB 9	Valeur numérique
	66BC 1A	
	66BD 0	

FV→	66BE	40	Type de variable : 4 (flottant) : nom sur une lettre (Ø+1)
	66BF	4D	M
	66C0	90	Valeur numérique
	66C1	9	
	66C2	1E	
	66C3	0	
	66C4	43	Type de variable : 4 (flottant); nom sur quatre lettres (3+1)
	66C5	44	D
	66C6	44	D
	66C7	44	D
	66C8	44	D
	66C9	0	
	66CA	37	Nombre d'octets occupés : $37 = 55 = 7 + (4 x 3 x 4)
	66CB	2	Nombre de dimensions (2)
	66CC	0	Nombre de variables dans la dimension 1
	66CD	3	Nombre de variables dans la dimension 2
	66CE	0	
	66CF	4	
	66D0	0	
	66D1	0	
FV+20→	66D2	0	

N.B. : les adresses données ici sont correctes si le lecteur de disques n'est pas branché. Les résultats obtenus sont différents avec le lecteur.

Remarques

Il existe trois pointeurs BASIC qui permettent de connaître l'emplacement de la zone des variables :

→ $611E - $611F : (VARDEB), pointe sur le début des variables numériques. DV=(PEEK(&H611E)*256+PEEK(&H611F))

→ $612Ø - $6121 : (VARFIN), pointe sur l'octet qui suit la fin de la dernière variable. FV= - - - -

→ $6122 - $6123 : (TABFIN), pointe sur l'octet qui suit la fin de dernier tableau.

Dans notre programme, au moment où la ligne 16 est exécutée, FV contient la valeur de VARFIN, c'est-à-dire $66BE. Pourtant, si l'on demande la valeur de VARFIN après l'exécution du programme, on obtient :

```
? HEX$(256*PEEK(&H6122)+PEEK(&H6123))
```

L'écran affiche 66C4 (si le disque n'est pas branché).

Ceci s'explique par le fait qu'entre-temps, une nouvelle variable a été déclarée : le M du compteur de la ligne 2Ø. La tableau DDDD a été déménagé, les variables indicées étant toujours placées après les variables simples.

Le premier octet du « champ » affecté à une variable contient deux informations : les huit bits de poids fort (en binaire) ou plutôt le chiffre de poids fort (en hexadécimal) est un code qui détermine le type de la variable :

2 : variable entière
3 : variable chaîne (les variables chaînes sont ailleurs, voir plus loin)
4 : variable simple précision
8 : variable double précision

Le chiffre de poids faible indique, à 1 près par défaut, le nombre de lettres utilisées pour définir le nom de la variable.

Pour les variables en tableau, on trouve, après la définition du type et du nom, deux octets qui forment un entier sur seize bits, qui représente le nombre d'octets mémoire utilisés par ce tableau. On trouve ensuite un octet qui donne le nombre de dimensions du tableau (un tableau ne peut pas avoir plus de seize dimensions). Enfin on trouve autant de couples d'octets que de dimensions qui donnent le nombre de variables dans cette dimension (la numérotation commençant à Ø, DIM C(3) réservera quatre variables). On constatera que les dimensions les plus à droite sont décrites en premier.

Le lecteur est invité à explorer un tableau afin de savoir dans quel ordre apparaissent ensuite les valeurs des différentes variables du tableau.

Le pointeur VARFIN ($612Ø - $6121) pointe sur le début de la définition du premier tableau déclaré. Le pointeur TABFIN ($6122 - $6123) pointe sur l'octet qui suit le dernier octet définissant le dernier tableau déclaré.

Après exécution du programme précédent on a :

```
? HEX$(256*PEEK(&H6120)+PEEK(&H6121))
```

L'écran affiche 67ØØ

Si l'on sait que les variables sont rangées en mémoire juste à la suite du programme BASIC, on comprendra que les modifications dans un programme, en modifiant l'emplacement de celui-ci, risquent « d'écraser » les variables. C'est en particulier cela qui explique qu'il soit impossible de continuer l'exécution d'un programme après l'avoir modifié, les variables ayant alors été « lessivées ».

Toutefois, il est possible de sauver l'espace des variables sous la forme d'un fichier binaire et de le restituer après l'avoir au besoin déplacé. Si l'on veut ensuite que tout marche bien il ne faudra pas oublier de remettre les pointeurs VARDEB, VARFIN, et TABFIN à leurs bonnes valeurs. C'est un jeu assez délicat mais qui peut avoir son utilité.

Les variables chaînes de caractères :

Elles sont tout aussi simplement accessibles que leurs collègues, les variables numériques, à l'aide de l'instruction VARPTR[1].

```
10 A$="TARTEMOLLE"
20 PRINT VARPTR(A$)
```

RUN donne 26 138. Cette valeur, ainsi que les suivantes, a été obtenue sans le lecteur de disques; elle peut être différente avec le lecteur.

Le descripteur de la chaîne A$ se trouve à cette adresse sur trois octets :

- le premier indique la longueur de la chaîne :

```
PRINT PEEK(26138)
```

L'écran affiche 1Ø

- les deux octets suivants donnent sur seize bits l'adresse du premier caractère de la chaîne :

```
PRINT 256*PEEK(26139)+PEEK(26140)
```

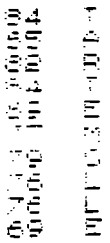
L'écran affiche 261Ø9

1.Le cas de l'instruction VARPTR est tout à fait exceptionnel : VARPTR accepte pour argument indifféremment une variable numérique ou une variable chaîne !

On peut alors retrouver les caractères de la chaîne :

```
FOR M=26109 TO 26119:D=PEEK(M):
PRINT D" "CHR$(D)
NEXT M
```

L'écran affiche



Le descripteur peut pointer à l'intérieur même du programme si la variable chaîne y est définie comme une constante (c'est le cas ici). Il peut également pointer vers une zone mémoire différente si la variable chaîne est le résultat d'un calcul.

Tout comme pour les variables numériques, le descripteur des variables chaînes se trouve dans la zone des variables située après le programme BASIC. Il est précédé d'un octet coupé en deux dont la moitié gauche est un « 3 » qui indique le « type chaîne », et dont la moitié droite indique, à 1 près par défaut, le nombre de caractères du nom de la variable. Suivent ensuite les octets contenant les codes ASCII du nom de la variable. C'est seulement après ces informations que l'on trouve les trois octets du descripteur.

En épluchant la zone variable du petit programme ci-dessus, on trouve :

3Ø Variable de type chaîne (3); nom sur une lettre (Ø).
41 Nom de la variable : A
ØE Longueur de la chaîne : 1Ø caractères
65 Adresse du premier caractère : 65FD

4Ø Variable numérique M
4D
8F
4C
CA
ØØ

```
10 A$="TARTEMOLLE"
20 PRINT HEX$(VARPTR(A$))
30 FOR M=&H65F0 TO &H65F0+110
33 PRINT HEX$(PEEK(M))" ";
40 NEXT M
```

```
RUN
6656
FF FF 0 0 0 54 9 0 48 41 24 D4 22 54 41
52 54 45 4D 4F 4C 4C 45 22 0 66 1A 0 14
AB 20 FF 95 28 FF 9E 28 41 24 29 29 0 66
 36 0 1E 81 20 4D D4 26 48 36 35 46 30
20 BB 20 26 48 36 35 46 30 C7 31 31 30 0
 66 4A 0 21 AB 20 FF 95 28 FF 8A 28 4D
29 29 22 20 22 3B 0 66 52 0 28 82 20 4D
0 0 0 30 41 A 65 FD 40 4D 8F 4C BA 0
```

Les variables alphanumériques obtenues par calcul (C$ = A$ + RIGHT$5B$,12)) sont dans une zone particulière. Cette zone réservée occupe a priori 256 octets, à moins que l'on en réserve plus (ou moins) au moyen du premier paramètre de l'instruction CLEAR.

Les pointeurs BASIC ($6128 - $6129) et ($612A - $612B) délimitent cette zone.

Interlude

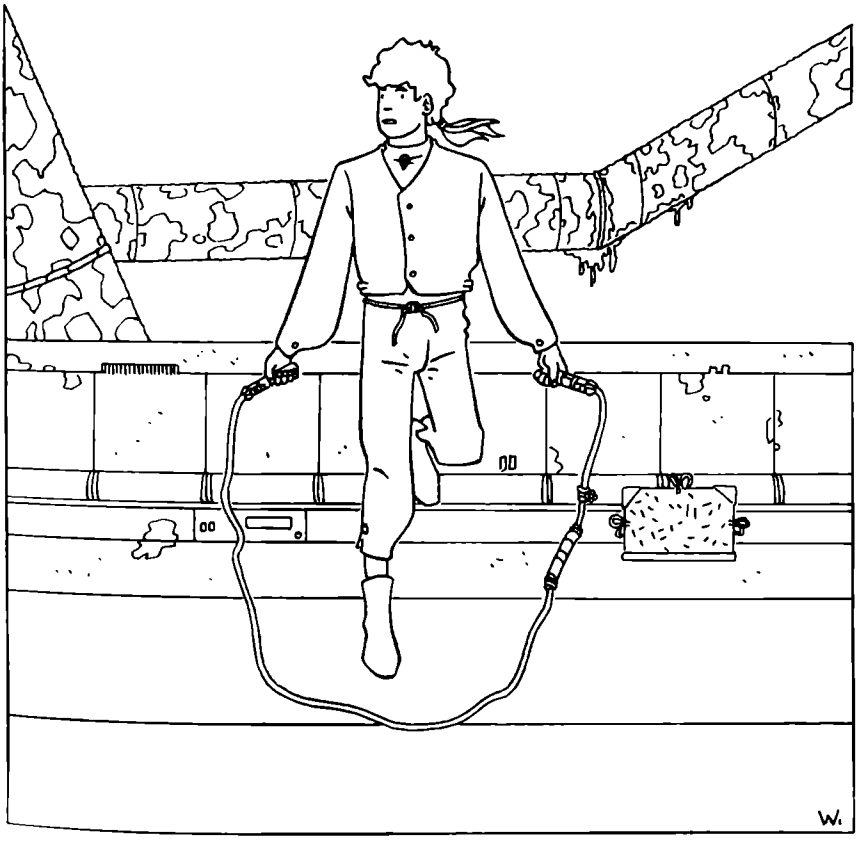
```
LIST

1 CLEAR .&H8000
2 '
5 '  Recherche de la pile S
6 '
7 '  10FF B100  STS $B100
8 '  39         RTS
9 '
10 DATA 10.FF.B1.00.39
20 FOR M=&HB000 TO &HB004
25 READ D$:POKE M.VAL("&H"+D$)
30 NEXT M
35 EXEC &HB000
40 S=256*PEEK(&HB100)+PEEK(&HB101)
50 PRINT HEX$(S)
OK

RUN
7ED0
OK
```

```
1 CLEAR .&HB000
2 '
5 '  Recherche de la pile S
6 '
7 '  10FF B100  STS $B100
8 '  39         RTS
9 '
10 DATA 10.FF.B1.00.39
20 FOR M=&HB000 TO &HB004
25 READ D$:POKE M.VAL("&H"+D$)
30 NEXT M
35 EXEC &HB000
40 S=256*PEEK(&HB100)+PEEK(&HB101)
50 PRINT HEX$(S)
OK

RUN
AED0
OK
```

Ce petit programme montre que BASIC place la pile S juste avant la plus haute mémoire disponible.

Question à 100 francs : à quoi sert la pile S ?

PEEK et POKE en comprenant



PEEK mémoire

Au chapitre précédent, nous avons entraperçu l'image dans la mémoire du programme que nous avons entré.

Voici un outil assez puissant d'analyse de la mémoire qui va nous permettre de bien comprendre comment tout cela s'organise. C'est le programme PEEK mémoire. Il est court mais il permet de travailler très efficacement et de comprendre beaucoup de choses. Quand on le lance, il demande une adresse de début. On peut donner cette adresse sous forme décimale (Ø) ou hexadécimale (&HØØØØ). Le programme donne le contenu des 160 mémoires dont les adresses suivent celle qui a été donnée. Sur chacune des vingt lignes, il y a le contenu de huit mémoires. L'adresse de la première est indiquée à gauche en hexadécimal. A droite, on obtient sur l'écran huit caractères qui représentent le contenu des huit mémoires. Ils sont en blanc si les codes sont inférieurs à 128 et en cyan si les codes sont supérieurs à 128. Ils sont sur fond bleu si les codes correspondent à des caractères de contrôle.

Une page de 160 caractères étant ainsi affichée, il est possible de la copier sur l'imprimante thermique 90-040 en tapant « C ». Il est possible de voir les 160 caractères suivants en tapant la barre d'espace ou ENTRÉE. Il est également possible de redonner une adresse de début en tapant « A ».

```
1 '          Peek memoire
2 '
10 CLS:SCREEN 7,0,0
20 INPUT "Adresse de debut :";A$
30 A=VAL(A$)
40 CLS:SCREEN 7,0,0:LOCATE 0,2
50 FOR M=A TO A+159 STEP 8
55 M$=HEX$(M)
60 IF LEN(M$)<4 THEN M$="0"+M$:GOTO 60
65 COLOR 7,0
66 PRINT M$;"  ";
70 D$=""
75 FOR M1=0 TO 7
80 D=PEEK(M+M1)
85 D$=D$+CHR$(D)
90 O$=HEX$(D)
91 IF LEN(O$)=1 THEN O$="0"+O$
```

```
92 PRINT O$;" ";
95 NEXT M1
100 FOR I=1 TO 8
105 D=ASC(MID$(D$,I,1))
110 COLOR 7,0
115 IF D>127 THEN D=D-128:COLOR 6
120 IF D<32 THEN D=D+64:COLOR ,4
125 PRINT CHR$(D);
130 NEXT I
140 PRINT
150 NEXT M
160 K$=INKEY$
161 K$=INKEY$:IF K$="" THEN 161
165 IF K$="A" THEN 10
166 IF K$="C" THEN SCREENPRINT
170 A=A+160:GOTO 40
```

C'est grâce à ce programme que j'ai pu localiser les pointeurs VARDEB, VARFIN, TABFIN ainsi que PROGDEB ($611C - $611D) qui donne l'adresse du début des programmes BASIC.

Voici quelques résultats obtenus avec ce programme.

```
0000   20 42 41 53 49 43 20 4D   BASIC M
0008   49 43 52 4F 53 4F 46 54  ICROSOFT
0010   20 31 2E 30 04 00 00 00   1.0D@@@
0018   00 00 60 FF 37 9B 37 9C  @@`■7[7\
0020   1C BE 1D 71 1C D5 0C 36  \>]q\UL6
0028   23 88 19 E0 23 FB 26 89  #H^`#{&I
0030   26 8F 26 DA 0F 15 0E 32  &O&ZOUN2
0038   0C 64 0E C3 0E 4F 0E 3B  LdNCNON;
0040   30 66 24 C3 25 2B 25 18  0f$C%+%X
0048   1D 61 11 64 11 65 36 17  ]a@d@e6W
0050   36 27 37 81 0E 5A 0E 77  6'7ANZNw
0058   0E 7E 10 58 17 EA 24 70  N~PXWj$p
0060   35 FD 30 99 35 F7 34 5A  5}0Y5w4Z
0068   34 62 30 D4 36 30 79 25  4b0T60y%
0070   90 79 25 82 7C 25 B6 7C  Py%B|%6|
0078   09 20 7F 26 6D 50 09 E5  I ■&mPIe
0080   46 09 EA 3C 09 EF 32 09  FIj<Io2I
0088   F4 28 09 F9 7A 26 5C 7B  t(Iyz&\{
0090   26 35 45 4E C4 46 4F D2  &5ENDFOR
0098   4E 45 58 D4 44 41 54 C1  NEXTDATA


00A0   44 49 CD 52 45 41 C4 4C  DIMREADL
00A8   45 D4 47 CF 52 55 CE 49  ETGORUNI
00B0   C6 52 45 53 54 4F 52 C5  FRESTORE
00B8   52 45 54 55 52 CE 52 45  RETURNRE
00C0   CD A7 53 54 4F D0 45 4C  M'STOPEL
00C8   53 C5 54 52 4F CE 54 52  SETRONTR
00D0   4F 46 C6 44 45 46 53 54  OFFDEFST
```

```
00D8   D2 44 45 46 49 4E D4 44  RDEFINTD
00E0   45 46 53 4E C7 44 45 46  EFSNGDEF
00E8   44 42 CC 4F CE 57 41 49  DBLONWAI
00F0   D4 45 52 52 4F D2 52 45  TERRORRE
00F8   53 55 4D C5 41 55 54 CF  SUMEAUTO
0100   44 45 4C 45 54 C5 4C 4F  DELETELO
0108   43 41 54 C5 43 4C D3 43  CATECLSC
0110   4F 4E 53 4F 4C C5 50 53  ONSOLEPS
0118   45 D4 4D 4F 54 4F D2 53  ETMOTORS
0120   4B 49 50 C6 45 58 45 C3  KIPFEXEC
0128   42 45 45 D0 43 4F 4C 4F  BEEPCOLO
0130   D2 4C 49 4E C5 42 4F D8  RLINEBOX
0138   55 4E 4D 41 53 CB 41 54  UNMASKAT

0140   54 52 C2 44 45 C6 50 4F  TRBDEFPO
0148   4B C5 50 52 49 4E D4 43  KEPRINTC
0150   4F 4E D4 4C 49 53 D4 43  ONTLISTC
0158   4C 45 41 D2 57 48 49 4C  LEARWHIL
0160   C5 57 45 4E C4 4E 45 D7  EWENDNEW
0168   53 41 56 C5 4C 4F 41 C4  SAVELOAD
0170   4D 45 52 47 C5 4F 50 45  MERGEOPE
0178   CE 43 4C 4F 53 C5 49 4E  NCLOSEIN
0180   50 45 CE 50 45 CE 50 4C  PENPENPL
0188   41 D9 54 41 42 A8 54 CF  AYTAB(TO
0190   53 55 C2 46 CE 53 50 43  SUBFNSPC
0198   A8 55 53 49 4E C7 55 53  (USINGUS
01A0   D2 45 52 CC 45 52 D2 4F  RERLERRO
01A8   46 C6 54 48 45 CE 4E 4F  FFTHENNO
01B0   D4 53 54 45 D0 AB AD AA  TSTEP+-*
01B8   AF DE 41 4E C4 4F D2 58  /↑ANDORX
01C0   4F D2 45 51 D6 49 4D D0  OREQVIMP
01C8   4D 4F C4 C0 BE BD BC 53  MOD@)=<S
01D0   47 CE 49 4E D4 41 42 D3  GNINTABS
01D8   46 52 C5 53 51 D2 4C 4F  FRESQRLO

01E0   C7 45 58 D0 43 4F D3 53  GEXPCOSS
01E8   49 CE 54 41 CE 50 45 45  INTANPEE
01F0   CB 4C 45 CE 53 54 52 A4  KLENSTR$
01F8   56 41 CC 41 53 C3 43 48  VALASCCH
0200   52 A4 45 4F C6 43 49 4E  R$EOFCIN
0208   D4 43 53 4E C7 43 44 42  TCSNGCDB
0210   CC 46 49 D8 48 45 58 A4  LFIXHEX$
0218   4F 43 54 A4 53 54 49 43  OCT$STIC
0220   CB 53 54 52 49 C7 47 52  KSTRIGGR
0228   A4 4C 45 46 54 A4 52 49  $LEFT$RI
0230   47 48 54 A4 4D 49 44 A4  GHT$MID$
0238   49 4E 53 54 D2 56 41 52  INSTRVAR
0240   50 54 D2 52 4E C4 49 4E  PTRRNDIN
0248   4B 45 59 A4 49 4E 50 55  KEY$INPU
0250   D4 43 53 52 4C 49 CE 50  TCSRLINP
0258   4F 49 4E D4 53 43 52 45  OINTSCRE
0260   45 CE 50 4F D3 50 54 52  ENPOSPTR
0268   49 C7 00 05 3B 15 78 16  IG@E;UXV
0270   05 06 63 0A 01 28 05 07  EFCJA(EG
0278   22 06 06 05 F2 06 97 05  "FFErFWE
```

```
0280  27 06 40 06 66 06 66 05  'F@F+F+E
0288  44 06 66 13 9E 13 9F 12  DF+3+3_R
0290  AC 12 AF 12 B2 12 B5 36  ,R/R2R56
0298  B5 0F 29 17 99 17 A4 13  5O)WYW$3
02A0  1D 12 F0 32 E2 35 EB 33  JRp2b5k3
02A8  53 34 EC 3A F8 39 C1 37  S41:x9A7
02B0  3F 35 E6 33 D5 34 F5 35  O5+3U4u5
02B8  07 34 53 33 16 18 01 0F  G4S3VXAO
02C0  1F 0F 78 05 6B 20 74 05  _OxEk.tE
02C8  7B 62 33 62 36 04 B6 2C  {b3b6D6.
02D0  B3 2E 99 2E 91 2E 0D 2D  3.Y.Q.M-
02D8  D9 36 E7 36 35 3F 14 4F  Y6g65?TO
02E0  4B 0D 0A 00 0D 0A 42 72  KMJ@MJBr
02E8  65 61 6B 00 20 45 72 72  eak@ Err
02F0  6F 72 00 30 64 9F 0F A6  or@0d_O&
02F8  84 C6 07 80 AF 27 12 C6  DFG@/'RF
0300  13 80 D2 26 15 AE 01 9F  S@R&U.A_
0308  11 9E 3F 27 09 9C 11 27  Q↑?'I\Q'
0310  09 9E 0F 3A 20 DF 9E 11  I↑O: _↑Q
0318  9F 3F 9E 0F 4D 39 8D 1A  _?↑OM9MZ
```

Adresse de début : &H0000. On est dans la ROM MEMO 7 BASIC. Les vingt-cinq premiers octets correspondent au titre de la cartouche qui apparaît à l'initialisation.

Viennent ensuite une centaine d'octets qui correspondent à des données en langage machine, puis apparaissent les mots connus du BASIC.

On pourra remarquer la manière dont le programmeur de chez Microsoft a séparé les mots pour économiser de la place : le dernier caractère d'un mot a son code ASCII augmenté de 128 (il apparaît en cyan). On économise ainsi un octet de séparation (, [virgule] ou; [point-virgule] entre chaque mot.

Adresse de début : &H6000. C'est le début de la zone RAM utilisateur. Les 256 premiers octets de cette zone sont les variables système utilisées par le moniteur.

Avec le disque

```
6000  FF FF FF FF FF FF FF FF  ████████
6008  FF FF FF FF FF FF FF FF  ████████
6010  FF FF FF FF FF FF FF FF  ████████
6018  FF 2F 60 05 60 00 60 18  ▼-E-@-X
6020  07 E8 44 E8 44 E8 44 31  GhDhDhD1
6028  23 00 00 00 00 BF FF E8  #@@@@?■h
6030  44 00 05 00 18 00 00 02  D@E@X@@B
```

```
6038   00 00 FF 38 00 00 00 00  @@█8@@@@
6040   00 00 00 00 00 00 00 07  @@@@@@@G
6048   02 00 00 11 0D 00 43 87  B@@QM@CG
6050   B2 00 11 80 00 80 00 80  2@Q@@@@@
6058   00 00 51 67 00 20 04 4A  @@Qg@ DJ
6060   40 00 5F 40 00 00 00 9A  @@_@@@@Z
6068   00 00 00 00 00 00 BE C1  @@@@@@>A
6070   00 00 00 00 00 00 00 00  @@@@@@@@
6078   00 F0 45 F0 58 F0 60 00  @pEpXp-@
6080   FF 00 00 00 00 00 00 00  █@@@@@@@
6088   00 00 00 00 00 00 00 00  @@@@@@@@
6090   00 00 00 00 00 00 00 00  @@@@@@@@
6098   00 00 00 00 00 00 00 00  @@@@@@@@


60A0   00 00 00 00 00 00 00 00  @@@@@@@@
60A8   00 00 00 00 00 00 00 00  @@@@@@@@
60B0   00 00 00 00 00 00 F2 04  @@@@@@rD
60B8   E7 D0 E2 25 E1 B6 58 E1  gPb%a6Xa
60C0   D1 E0 A8 06 59 01 04 60  Q-(FYAD-
60C8   86 F4 00 04 60 48 62 37  Ft@D-Hb7
60D0   00 27 02 80 7F 80 7F 80  @'B@█@█@
60D8   7F 80 7F 80 7F 80 7F 80  █@█@█@█@
60E0   6B 80 7F 80 7F 80 7F 80  k@█@█@█@
60E8   7F 80 7F 80 7F 80 7F 80  █@█@█@█@
60F0   7F 80 7F 80 7F 80 7F 80  █@█@█@█@
60F8   7F 80 7F 80 7F 80 FF 00  █@█@█@█@
6100   00 22 00 00 00 04 00 00  @"@@@D@@
6108   00 00 00 65 8A 65 87 BE  @@@eJeG>
6110   B6 8A 70 00 00 00 00 0A  6Jp@@@@J
6118   00 00 BE 8E 88 33 8A 5C  @@>NH3J\
6120   8A 94 8A 94 BE D2 BF 6F  JTJT>R?o
6128   BF 5F BF FE 00 50 00 A1  ?_?¯@P@!
6130   00 28 00 00 89 1F 00 00  @(@@I_@@
6138   88 32 64 4B 02 8A 70 8A  H2dKBJpJ


6140   86 79 04 00 8A 94 BE D4  FyD@JT>T
6148   00 00 00 00 00 00 65 8A  @@@@@@eJ
6150   00 64 44 88 85 8F 00 61  @dDHEO@a
6158   58 00 00 00 00 42 00 01  X@@@@B@A
6160   89 71 00 8F C2 C0 00 00  Iq@OB@@@
6168   00 00 00 42 42 00 89 73  @@@BB@Is
6170   00 00 00 00 00 00 00 00  @@@@@@@@
6178   89 7B 00 00 00 00 00 8A  I{@@@@@J
6180   37 8A 49 00 A6 00 00 00  7JI@&@@@
6188   00 00 00 00 BE AB 00 00  @@@@>+@@
6190   00 00 00 89 2D 00 00 00  @@@I-@@@
6198   00 00 00 00 00 00 0D 1A  @@@@@@MZ
61A0   04 28 00 00 19 00 EB 1A  D(@@Y@kZ
61A8   00 18 0D 00 00 00 00 00  @XM@@@@@
61B0   00 00 0C BA 26 02 0C B9  @@L:&BL9
61B8   B6 89 2E 81 3A 24 0A 81  6I.A:$JA
61C0   20 26 02 0E B2 80 30 80   &BN2@0@
61C8   D0 39 7E 07 E9 7E 25 02  P9¯Gi¯%B
61D0   7E 07 EB 00 00 00 00 00  ¯Gk@@@@@
61D8   00 00 00 00 00 00 00 00  @@@@@@@@
```

```
6000  FF FF FF FF FF FF FF FF  ████████
6008  FF FF FF FF FF FF FF FF  ████████
6010  FF FF FF FF FF FF FF FF  ████████
6018  FF 2F 60 05 60 00 60 18  ▒/`.`.`.
6020  07 E8 44 E8 44 E8 44 31  GhDhDhD1
6028  23 10 10 00 00 BF FF E8  #PP@@?▒h
6030  44 00 05 00 18 00 00 02  D@E@X@@B
6038  00 00 FF 38 00 00 00 00  @@▒8@@@@
6040  00 00 00 00 00 00 00 07  @@@@@@@G
6048  02 00 00 00 01 00 10 62  B@@@A@Pb
6050  00 80 00 80 00 80 00 80  @@@@@@@@
6058  00 00 51 67 00 20 04 49  @@Qg@ DI
6060  40 00 5F 40 00 00 00 9A  @@_@@@@Z
6068  00 00 00 00 00 00 BE C1  @@@@@@@)A
6070  00 00 00 00 00 00 00 00  @@@@@@@@
6078  00 F0 45 F0 58 F0 60 FF  @pEpXp`▒
6080  00 00 00 00 00 00 00 00  @@@@@@@@
6088  00 00 00 00 00 00 00 00  @@@@@@@@
6090  00 00 00 00 00 00 00 00  @@@@@@@@
6098  00 00 00 00 00 00 00 00  @@@@@@@@

60A0  00 00 00 00 00 00 00 00  @@@@@@@@
60A8  00 00 00 00 00 00 00 00  @@@@@@@@
60B0  00 00 00 00 00 00 F2 8D  @@@@@@@r▒
60B8  B3 E7 01 E7 D0 E1 64 E1  3gAgPadá
60C0  AA E0 A8 06 50 02 6D 60  *`(.Pm`
60C8  62 00 62 7F E0 C9 E0 47  b@b▒`I`G
60D0  00 FF 02 00 FF 00 FF 00  @▒@@▒@▒@
60D8  FF 00 FF 00 FF 00 FF 00  ▒@▒@▒@▒@
60E0  FB 00 FF 00 FF 00 FF 00  {@▒@▒@▒@
60E8  FF 00 FF 00 FF 00 FF 00  ▒@▒@▒@▒@
60F0  FF 00 FF 00 FF 00 FF 00  ▒@▒@▒@▒@
60F8  FF 00 FF 00 FF 00 FF 40  ▒@▒@▒@▒@
6100  00 22 00 00 00 04 00 00  @"@@@D@@
6108  00 00 00 65 8A 65 87 BE  @@@eŠe‡@
6110  B6 68 32 00 00 00 00 0A  ¶h2@@@@J
6118  00 00 BE 8E 65 F5 68 1E  @@)Neuh†
6120  68 51 68 51 BE D2 BE E2  hQhQ)R)b
6128  BF D2 BF FE 00 50 00 5B  ?R?_@P@[
6130  66 49 00 00 66 E1 00 00  fI@@fá@@
6138  65 F4 64 4B 02 68 32 68  etdKBh2h

6140  48 79 04 00 68 56 68 51  HyD@hVhQ
6148  00 00 00 00 00 00 65 8A  @@@@@@eŠ
6150  00 64 44 66 47 8F 00 61  @dDfGO@a
6158  58 00 00 00 00 42 00 01  X@@@@B@A
6160  66 33 00 8F C2 C0 00 00  f3@OB@@@
6168  00 00 00 42 42 00 67 35  @@@BB@g5
6170  00 00 00 00 00 00 00 00  @@@@@@@@
6178  67 3D 00 00 00 00 00 00  g=@@@@@@
6180  00 00 00 00 00 00 00 00  @@@@@@@@
6188  00 00 00 00 BE AB 00 00  @@@@)+@@
```

```
6190  00 00 00 66 EF 00 00 00 @@@f○@@@@
6198  00 00 00 00 00 00 0D 1A @@@@@@@MZ
61A0  04 28 00 00 19 00 EB 1A D(@@Y@kZ
61A8  00 18 0D 00 00 00 00 00 @XM@@@@@@
61B0  00 00 0C BA 26 02 0C B9 @@L:&BL9
61B8  B6 66 F0 81 3A 24 0A 81 6fpA:$JA
61C0  20 26 02 0E B2 80 30 80  &BN2@0@
61C8  D0 39 7E 07 E9 7E 25 02 P9~Gi~%B
61D0  7E 07 EB 00 00 00 00 00 ~Gk@@@@@@
61D8  00 00 00 00 00 00 00 00 @@@@@@@@@
```

Après ces zones on trouve les mémoires tampons d'entrée-sortie par lesquelles se font, par exemple, les échanges entre la mémoire centrale du micro-ordinateur et le lecteur de cassettes ou de disquettes. Avec de l'expérience et de la patience, on pourra dresser une cartographie assez précise de la zone mémoire commençant à $6ØØØ.

Dans la version cassette, cette zone va jusqu'en &H65F4, où commence le programme BASIC. Dans la version disque, on trouve le début du programme BASIC en &H8832, les mémoires comprises entre &H6ØØØ et &H88ØØ étant alors occupées par le DOS (« disk operating system ») qui gère tous les mots propres à la gestion du disque tels DIR, DSKINI, etc., et qui occupe plus de 8 K.

Où est le programme BASIC ?

Il est intéressant d'étudier l'image, dans la mémoire, du programme BASIC. On trouve tout d'abord un octet à Ø, puis deux octets qui pointent vers le début de la ligne BASIC suivante (ici, 88 51 en version disque). On trouve ensuite deux octets qui correspondent au numéro de la ligne (ici ØØ Ø1 : ligne 1). Suivent ensuite les codes correspondant à la ligne de programme.

```
8837  3A   : séparateur d'instruction
8838  8D   ' REM
8839  20   (espace) Huit espaces
883A  20
883B  20
883C  20
883D  20
883E  20
```

```
383F   20
3840   20
3841   20

3842   50   P
3843   65   E
3844   65   E
3845   6B   K
3846   20   (espace)
3847   6D   m
3848   16   ] codes de l'accent aigu
3849   42   ]
384A   65   e
384B   6D   m
384C   6F   o
384D   69   i
384E   72   r
384F   65   e
3850   00   code de fin de ligne

3851   88   adresse de la ligne suivante
3852   58
3853   00   ligne de programme n° . 2
3854   02
3855   3A   :
3856   8D   ,
3857   00   code de fin de ligne

3858   88   adresse de la ligne suivante
3859   67
385A   00   ligne de programme n° . 1Ø
385B   0A
385C   9D   code pour CLS
385D   3A   :

385E   FF   code pour SCREEN
385F   A4
3860   20   (espace)
3861   37   7
3862   2C   /
3863   30   Ø
3864   2C   /
3865   30   Ø
3866   00   code de fin de ligne

3867   88   adresse de la ligne suivante
3868   88
3869   00   ligne de programme n° . 2Ø
386A   14
386B   FF   code pour INPUT
```

```
886C   A1
886D   20   (espace)
886E   22   "
886F   41   A
8870   64   d
8871   72   r
8872   65   e
8873   73   s
8874   73   s
8875   65   e
8876   20
8877   64   d
8878   65   e
8879   20
887A   64   d
887B   16
887C   42   B
887D   65   e
887E   62   b
887F   75   u
8880   74   t
8881   20
8882   3A   :
8883   22   "
8884   3B   ;
8885   41   A
8886   24   $
8887   00

8888   88
8889   95
888A   00   Code de fin de ligne
```

On remarque que les instructions BASIC sont codées sur un ou deux octets de codes supérieurs à 128. Le reste est codé de manière naturelle en ASCII.

Les mots du BASIC

Voici la liste des mots BASIC avec leurs codes et le programme qui a permis de les trouver (je n'ai pas inclus dans ce programme le listage des mots DOS; je suis sûr que le lecteur intéressé ne manquera pas de compléter cette liste).

Le programme :

```
1 '       Mots du BASIC
2 '
3 OPEN "O",#1,"LPRT:
4 CLEAR 4096
5 DIM PT%(125),MOT$(125),COD$(125)
10 CLS:SCREEN 7,0,0
15 M=&H92
20 I=0
30 L=PEEK(M)
35 IF L>127 THEN 50
40 MOT$(I)=MOT$(I)+CHR$(L)
45 M=M+1:GOTO 30
50 L=L-128
55 MOT$(I)=MOT$(I)+CHR$(L)
100 C=I+128
101 IF C>213 THEN C=65280+C-86
102 C$=HEX$(C)
103 IF C<256 THEN C$="  "+C$
105 COD$(I)=C$
106 PT%(I)=I
110 PRINT#1, COD$(I);" ";MOT$(I)
120 M=M+1:I=I+1
130 IF I<125 THEN 30
140 '
141 '  Tri alphabetique
142 '
150 EX=0
160 FOR I=0 TO 123
165 IF MOT$(I)<=MOT$(I+1) THEN 180
170 SWAP MOT$(I),MOT$(I+1)
171 SWAP COD$(I),COD$(I+1)
175 EX=1
180 NEXT I
181 FOR I=123 TO 0 STEP-1
182 IF MOT$(I)<=MOT$(I+1) THEN 185
183 SWAP MOT$(I),MOT$(I+1):SWAP COD$(I),
COD$(I+1)
184 EX=1
185 NEXT I
190 IF EX=1 THEN 150
200 PRINT:PRINT
210 FOR I=0 TO 124
211 PRINT#1, MOT$(PT%(I)),COD$(PT%(I))
212 NEXT I
213 CLOSE
```

Les résultats : Les mots réservés du BASIC

Classés par ordre des codes

80 END	AB PRINT	FF80 SGN
81 FOR	AC CONT	FF81 INT
82 NEXT	AD LIST	FF82 ABS
83 DATA	AE CLEAR	FF83 FRE
84 DIM	AF WHILE	FF84 SQR
85 READ	B0 WEND	FF85 LOG
86 LET	B1 NEW	FF86 EXP
87 GO	B2 SAVE	FF87 COS
88 RUN	B3 LOAD	FF88 SIN
89 IF	B4 MERGE	FF89 TAN
8A RESTORE	B5 OPEN	FF8A PEEK
8B RETURN	B6 CLOSE	FF8B LEN
8C REM	B7 INPEN	FF8C STR$
8D '	B8 FEN	FF8D VAL
8E STOP	B9 PLAY	FF8E ASC
8F ELSE	BA TAB(	FF8F CHR$
90 TRON	BB TO	FF90 EOF
91 TROFF	BC SUB	FF91 CINT
92 DEFSTR	BD FN	FF92 CSNG
93 DEFINT	BE SPC(	FF93 CDBL
94 DEFSNG	BF USING	FF94 FIX
95 DEFDBL	C0 USR	FF95 HEX$
96 ON	C1 ERL	FF96 OCT$
97 WAIT	C2 ERR	FF97 STICK
98 ERROR	C3 OFF	FF98 STRIG
99 RESUME	C4 THEN	FF99 GR$
9A AUTO	C5 NOT	FF9A LEFT$
9B DELETE	C6 STEP	FF9B RIGHT$
9C LOCATE	C7 +	FF9C MID$
9D CLS	C8 -	FF9D INSTR
9E CONSOLE	C9 *	FF9E VARPTR
9F PSET	CA /	FF9F RND
A0 MOTOR	CB ^	FFA0 INKEY$
A1 SKIPF	CC AND	FFA1 INPUT
A2 EXEC	CD OR	FFA2 CSRLIN
A3 BEEP	CE XOR	FFA3 POINT
A4 COLOR	CF EQV	FFA4 SCREEN
A5 LINE	D0 IMP	FFA5 POS
A6 BOX	D1 MOD	FFA6 PTRIG
A7 UNMASK	D2 @	
A8 ATTRB	D3 >	
A9 DEF	D4 =	
AA POKE	D5 <	

Classés par ordre alphabétique

'	8D
*	C9
+	C7
-	C8
/	CA
<	D5
=	D4
>	D3
@	D2
ABS	FF82
AND	CC
ASC	FF8E
ATTRB	A8
AUTO	9A
BEEP	A3
BOX	A6
CDBL	FF93
CHR$	FF8F
CINT	FF91
CLEAR	AE
CLOSE	B6
CLS	9D
COLOR	A4
CONSOLE	9E
CONT	AC
COS	FF87
CSNG	FF92
CSRLIN	FFA2
DATA	83
DEF	A9
DEFDBL	95
DEFINT	93
DEFSNG	94
DEFSTR	92
DELETE	9B
DIM	84
ELSE	8F
END	80
EOF	FF90
EQV	CF
ERL	C1
ERR	C2
ERROR	98
EXEC	A2
EXP	FF86
FIX	FF94
FN	BD
FOR	81
FRE	FF83
GO	87
GR$	FF99
HEX$	FF95
IF	89
IMP	D0
INKEY$	FFA0
INPEN	B7
INPUT	FFA1
INSTR	FF9D
INT	FF81
LEFT$	FF9A
LEN	FF8B
LET	86
LINE	A5
LIST	AD
LOAD	B3
LOCATE	9C
LOG	FF85
MERGE	B4
MID$	FF9C
MOD	D1
MOTOR	A0
NEW	B1
NEXT	82
NOT	C5
OCT$	FF96
OFF	C3
ON	96
OPEN	B5
OR	CD
PEEK	FF8A
PEN	B8
PLAY	B9
POINT	FFA3
POKE	AA
POS	FFA5
PRINT	AB
PSET	9F
PTRIG	FFA6
READ	85
REM	8C
RESTORE	8A
RESUME	99
RETURN	8B
RIGHT$	FF9B
RND	FF9F
RUN	88
SAVE	B2
SCREEN	FFA4
SGN	FF80
SIN	FF88
SKIPF	A1
SPC(	BE
SQR	FF84
STEP	C6
STICK	FF97
STOP	8E
STR$	FF8C
STRIG	FF98
SUB	BC
TAB(	BA
TAN	FF89
THEN	C4
TO	BB
TROFF	91
TRON	90
UNMASK	A7
USING	BF
USR	C0
VAL	FF8D
VARPTR	FF9E
WAIT	97
WEND	B0
WHILE	AF
XOR	CE
^	CB

Quand on étudie un programme court, on se rend compte que BASIC sait que le programme est terminé quand le pointeur d'adresse de la ligne suivante pointe sur ØØ ØØ. On peut donc repérer dans la mémoire la fin d'un programme en partant à la recherche de trois octets consécutifs à Ø (le premier pour la fin de la dernière ligne du programme et les deux autres comme pointeurs de fin de programme). Plus simplement, on peut utiliser les pointeurs PROGDEB ($611C - $611D) et VARDEB ($611E - $611F) pour déterminer les adresses de début et de fin des programmes. La connaissance de cette structure permet de réaliser tout un tas de belles choses.

Trucages divers

Ce petit programme va nous permettre de réaliser quelques trucages.

```
10 '     Ceçi est une démonstration.
20 PLAY "L10MIREDOSIL20LA"
30 END
```

Tout d'abord, on étudie bien la structure binaire grâce à une instruction tapée en mode immédiat :

```
FOR M=&H65F4 TO &H65F4+60:?HEX$(PEEK(M))
" ";:NEXT M
```

Cette instruction ne fait pas partie du programme mais elle permet d'en voir l'image binaire :

Adresse	Code hexadécimal	Signification
65F4	0	Début du programme
65F5	66 1F	Lien vers la ligne suivante
65F7	0 A	Ligne de programme n°1Ø (A)
65F9	3A	:
65FA	8D	'
65FB	20	(espace)
65FC	20	...code ASCII du message de la
65FD	20	ligne 1Ø5. Remarquez le codage
65FE	20	sur trois octets du «ç» (16, 4B,
65FF	43	63) et du «é» (16,42, 65).
6600	65	

```
6601    16
6602    4B
6603    63
6604    69
6605    20
  :
661C    6E
661D    2E
661E    0       Fin de la ligne
661F    66      Lien vers la ligne suivante
6620    38      Ligne de programme n° . 2Ø (14)
6621    0
6622    14
6623    B9      PLAY
6624    20      (espace)
6625    22      «
6626    4C
                ...musique...
  :
6634    4C
6635    41
6636    22      »
6637    0       Fin de ligne
6638    66      Lien vers la ligne suivante
6639    3E
663A    0       Ligne du programme n° . 3Ø (IE)
663B    1E
663C    80      END
663D    0       Fin de ligne
663E    0
663F    0       Fin du programme
6640    40      Variable M
6641    4D
6642    8F
6643    4C
6644    88
```

Pour que nos expériences marchent bien, il est nécessaire de taper le programme exactement comme il est écrit ci-dessus, en respectant tous les espaces, etc., faute de quoi les adresses risquent de varier. Dans ce cas, le lecteur saura sans doute malgré tout s'y retrouver et rectifier ce qui doit l'être. D'ailleurs s'il veut tirer parti de ces trucs pour ses propres programmes, il faudra bien qu'il se débrouille pour adapter ces idées à ses besoins.

La ligne fantôme

POKE &H65F6, &H38 indique que la deuxième ligne de programme est en fait la ligne 3Ø.
LIST montre effectivement que la ligne 2Ø a disparu.
Pourtant RUN montre que le programme exécute cette ligne.

```
POKE &H65F6,&H38

LIST

10 '     Ceçi est une démonstration.
30 END

RUN

POKE &H65F6,&H1F

LIST

10 '     Ceçi est une démonstration.
20 PLAY "L10MIREDOSIL20LA"
30 END
```

Il est possible de camoufler de cette manière des parties de listing pour y mettre diverses choses : bouts de programmes assembleur ou signature dans une ligne de remarque fantôme ou encore une ligne qui produira un effet inattendu.

```
TOTO

OK
LIST

10 CLS:SCREEN 3,4,4
20 LOCATE 10,10
30 ATTRB 1,1
40 PRINT "TATA"
50 ATTRB 0,0
70 END
OK
```

Comment ce programme peut-il bien donner ce résultat ?

Tout s'explique :

```
10 CLS:SCREEN 3,4,4
20 LOCATE 10,10
30 ATTRB 1,1
32 PRINT "TOTO":GOTO 50
40 PRINT "TATA"
50 ATTRB 0,0
70 END
OK
```

```
FOR M=&H65F4 TO &H65F4+100:?HEX$(PEEK(M)
);" ":NEXT
0 66 4 0 A 9D 3A FF A4 20 33 2C 34 2C 34
 0 66 10 0 14 9C 20 31 30 2C 31 30 0 66
1A 0 1E A8 20 31 2C 31 0 66 2D 0 20 AB
20 22 54 4F 54 4F 22 3A 87 BB 20 35 30 0
 66 3A 0 28 AB 20 22 54 41 54 41 22 0 66
 44 0 32 A8 20 30 2C 30 0 66 4A 0 46 80
0 0 0 40 4D 8F 4C A0 0 0 0 FF 0 FF 0 FF
OK
POKE&H6605,&H1A
OK
```

Voici en fait le programme...dont on a camouflé la ligne 32 !

Le listage fou

```
LIST

10 '     Ceci est une démonstration.
20 PLAY "L10MIREDOSIL20LA"
30 END
OK

POKE &H663E,&H65
POKE &H663F,&HF4
```

Le programme marche comme prévu mais LIST plonge l'ordinateur dans la perplexité la plus complète. Le lecteur saura sûrement tirer parti de ces informations pour se livrer à des expériences personnelles tout à fait passionnantes.

Le programme miraculé

Votre programme est là, en mémoire.

```
LIST

10 '     Ceci est une démonstration.
20 PLAY "L10MIREDOSIL20LA"
30 END

NEW

LIST

OK
```

Comme on peut s'y attendre, le programme a disparu. LIST renvoie l'image d'une mémoire vide et RUN n'a aucun effet.

```
PRINT PEEK(&H65F5),PEEK(&H65F6)
 0             0
```

Quelques PEEK aux adresses &H65F5 et suivantes permettent de constater que NEW a seulement modifié les deux octets &H65F5 et &H65F6 qui ont été mis à Ø. Le reste est là, intact.
Attention : dans cette exploration prenez soin de ne pas utiliser de variables.
En particulier, ne tapez pas FOR M = &H65F5 TO &H65F5 + 1ØØ : PRINT PEEK(M) : NEXT M, car la variable (M) viendrait se placer en fin de programme, c'est-à-dire aux adresses &H65F5 et suivantes, écrasant ainsi ce que nous allons essayer de sauver.
Pour retrouver le programme, il suffit de refaire le lien.

```
POKE &H65F5,&H66:POKE &H65F6,&H1F

LIST

10 '     Ceci est une démonstration.
20 PLAY "L10MIREDOSIL20LA"
30 END
OK
```

On éprouve incontestablement un sentiment de bonheur en retrouvant le programme en ordre de marche.
Ceci peut être bien utile pour récupérer un programme perdu : à coups de PEEK, sans utiliser de variables, on recherche la fin de la première ligne (le Ø suivi du lien et du numéro de la deuxième ligne), on rétablit alors le lien de la première ligne aux adresses &H66F5 et &H66F6 pour qu'il pointe sur le lien de la deuxième ligne. Le programme peut alors être listé. En général, il ne peut pas être sauvé tel que, l'ordinateur croyant encore quelque part que la longueur du programme est Ø, mais en le listant sur cassette ou en le sauvant en ASCII avec l'option A de SAVE, on peut s'en tirer. On peut également partir à la recherche des trois zéros qui signalent la fin du programme, puis, les ayant trouvés, rétablir le pointeur VARDEB à sa juste valeur.
Sauriez-vous à présent récupérer un programme sauvé avec l'instruction P [1] ? Ou encore, retrouver un programme alors que l'initialisation semble indiquer que tout est perdu ?

1.En fait, il y a beaucoup plus simple pour cela :
Initialisation
EXEC 1216
Initialisation
LIST
SAVE" ",A
RUN

La valse des numéros de ligne

Une autre sorte de gags permise par cette étude consiste à perturber les numéros de lignes de l'extérieur. Par exemple :

```
10 '     Ceçi est une démonstration.
20 PLAY "L10MIREDOSIL20LA"
30 END

POKE &H65F8,30
POKE &H663B,10

LIST

30 '     Ceçi est une démonstration.
20 PLAY "L10MIREDOSIL20LA"
10 END
```

Curieux numéros de lignes ? ? ?

Tout s'explique...

Le programme continue de marcher très bien mais il peut surprendre ! On peut également donner le même numéro à toutes les lignes mais les GOTO et les GOSUB risquent de s'y perdre quelque peu. Il n'y a pas de réajustement de l'ordre des lignes, car celui-ci est fait au moment où vous validez votre ligne par ENTREE. Quand on intervient dans la mémoire, on intervient après que l'interpréteur BASIC a fait son travail d'édition.

Une dernière opération amusante consiste à introduire des caractères de contrôle dans les lignes de remarques. En utilisant le fait que certains caractères, quand on les imprime, produisent certains effets sur l'écran, il est possible de bricoler des programmes dont le listage peut surprendre.

```
10 '        PROGRAMME INVISIBLE  XXXX
20 '
30 '        Ici la suite du programme
40 '        ....
50 '        ....
60 '        ....
70 '
80 '
90 '        FIN X
OK
```

```
FOR M=&H65F0 TO &H65F0+128 STEP 8:PRINT
HEX$(M);" - ";:FOR M1=0 TO 7:?HEX$(PEEK
(M+M1));" ";:NEXTM1:PRINT:NEXT M
65F0 - FF FF 0 0 0 66 1B 0  2E
65F8 - A 3A 8D 20 20 20 20 20
6600 - 20 50 52 4F 47 52 41 4D
6608 - 4D 45 20 49 4E 56 49 53
6610 - 49 42 4C 45 20 20 58 58
6618 - 58 58 0 66 22 0 14 3A
6620 - 8D 0 66 48 0 1E 3A 8D
6628 - 20 20 20 20 20 20 49 63
6630 - 69 20 6C 61 20 73 75 69
6638 - 74 65 20 64 75 20 70 72
6640 - 6F 67 72 61 6D 6D 65 0
6648 - 66 59 0 28 3A 8D 20 20
6650 - 20 20 20 20 2E 2E 2E 2E
6658 - 0 66 6A 0 32 3A 8D 20
6660 - 20 20 20 20 20 2E 2E 2E
6668 - 2E 0 66 7B 0 3C 3A 8D
6670 - 20 20 20 20 20 20 2E 2E
OK

POKE &H6617,&H1B
POKE &H6618,&H41
POKE &H6619,&H1B
POKE &H661A,&H51
POKE &H6699,&H0C
```

Si on essaie de lister le programme, l'écran devient tout rouge et l'on ne voit plus rien ! Même un SCREEN 7,Ø,Ø ne fait rien paraître ! LIST « LPRT : donne tout de même quelques précisions mais la ligne 9Ø a quand même disparu et la ligne 1Ø semble avoir été modifiée.

```
10 '       PROGRAMME INVISIBLE  XAQf"
20 '
30 '       Ici la suite du programme
40 '       ....
50 '       ....
60 '       ....
70 '
80 '
```

J'ai utilisé (rarement) cette technique pour faire ressortir en couleurs des portions intéressantes de listage. Un petit utilitaire que je merge en 1ØØØØ me permet de remplacer les séquences de deux @ (aérobas) par des séquences ESCAPE ou par des codes RAZ.

Interlude

Pour écrire sur deux lignes ce qui était écrit sur une seule, on a ceci :

```
10 SCREEN 3,4,4:CLS
```

On veut :

```
10 SCREEN 3,4,4
15 CLS
```

Vous l'auriez trouvé tout seul, il suffit d'amener le curseur sur les deux points qui séparent les deux instructions. Puis, à l'aide de la touche INS, on insère autant de blancs qu'il en faut pour que le CLS vienne sur la ligne suivante à l'aplomb de SCREEN. On tape alors Xc[1] qui efface la fin de la ligne 10 et le lien de cette ligne de programme avec ce qui pourrait se trouver sur les lignes d'écran suivantes (en particulier le : CLS). On valide cette ligne 10 diminuée, et le curseur tombe pile là où il faut pour écrire le 15 et d'un coup de blanc supprimer les « : ». On valide la ligne 15 et le tour est joué.

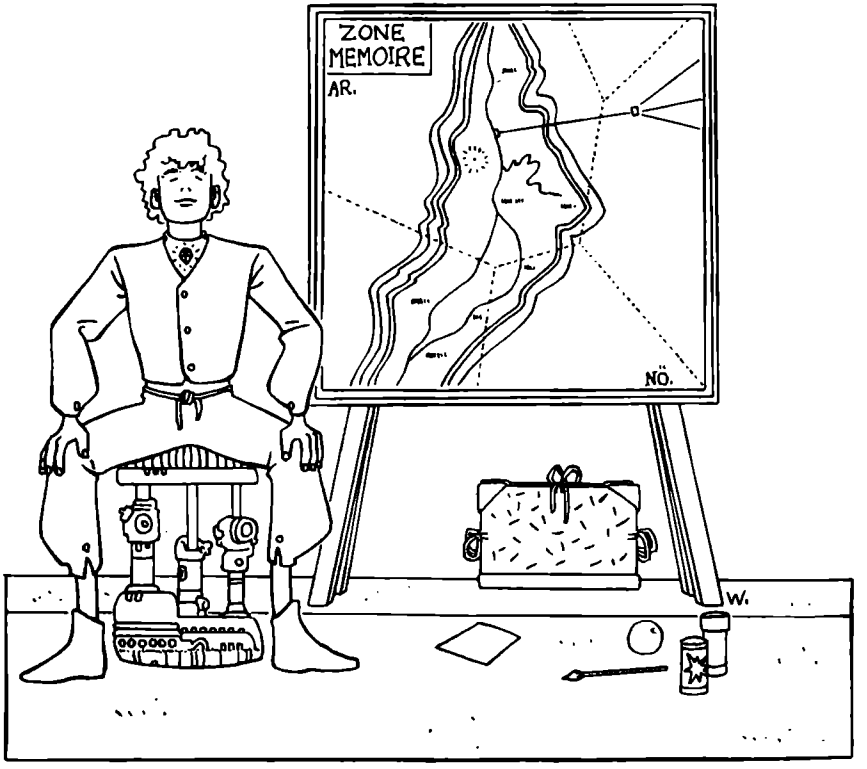
Plus dur : pour ne faire qu'une ligne de ce qui était écrit sur deux.

Il faut commencer par poser un « lien » en fin de ligne d'écran signalant que la ligne de programme se poursuit sur la ligne d'écran suivante. Pour ce faire, voici un truc trouvé par J.-P. Duclos : on place le curseur en fin de ligne (en colonne 39), on tape deux fois ACC, ce qui ne fait pas avancer le curseur, puis une fois sur la barre d'espace. Le curseur passe alors à la ligne suivante sans ouvrir de place pour l'insertion. On peut alors déplacer le curseur en marche arrière et utiliser EFF pour supprimer les blancs qui seraient de trop.

Sur TO7-70, on peut facilement poser un lien en tapant Wc[1] en fin de ligne. Dans la machine, la présence d'un lien en fin de ligne I est indiquée par un 0 à l'adresse $H6000 + I.
Un bug : essayez de taper une séquence ESCAPE quand le curseur est sur la dernière colonne de la dernière ligne...

1. Xc est une notation de CNT-X.

Codes ASCII et normes vidéotex



Le lecteur connaît sûrement le code ASCII : dans le chapitre précédent, nous avons rencontré des lettres connues par la machine sous la forme de leurs codes. Ces codes peuvent paraître assez arbitraires; pourtant, leur interprétation permet de belles choses.

La routine PUTCH

Toute la gestion de l'écran est faite par le moniteur du TO7 par l'intermédiaire de la routine PUTCH qui réside à l'adresse E8Ø3 où passe le caractère à envoyer à PUTCH par l'intermédiaire de l'accumulateur B.

Le programme assembleur suivant :

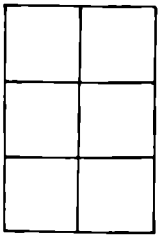
```
LDB # 12
JMP $E8Ø3
```

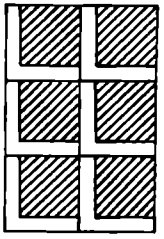
est ainsi équivalent au programme BASIC suivant :

```
PRINT CHR$(12);
```

Selon ses auteurs, la routine PUTCH est un interpréteur vidéotex. Ceci est largement exact dans la mesure où les séquences de caractères envoyées à PUTCH produisent à peu près toujours sur l'écran du TO7 les mêmes effets que si elles étaient envoyées à un Minitel. Il existe toutefois quelques différences :

- en vidéotex, les caractères sont définis sur une matrice 8 x 10 et non 8 x 8. Ceci permet en particulier de réaliser des caractères semi-graphiques « soulignés » qui n'existent pas sur le TO7 :





- le TO7 ne reconnaît pas les caractères clignotants

- en vidéotex, un retour à la ligne (CHR$(13)) remet les attributs de taille des caractères à leurs valeurs par défaut (ATTRB Ø,Ø) ce qui n'est pas le cas avec PUTCH.

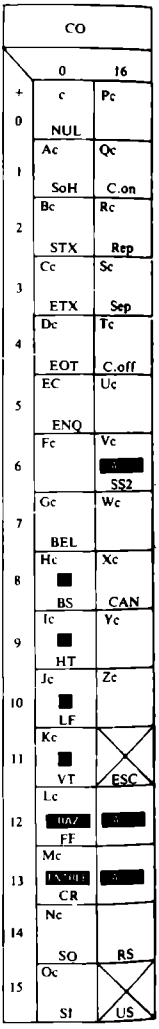
Mais à part ces différences (bugs ?), les codes de contrôle et les séquences ESCAPE sont interprétés correctement.

PUTCH connaît les diverses « grilles » :

CØ : 20 commandes (caractères de contrôle) ($Ø - $1F)
C1 : 38 codes d'attribut ($4Ø - $6F)
GØ : 96 caractères affichables ($2Ø - $7F)
G1 : 64 caractères semi-graphiques ($2Ø - $7F)
G2 : 20 caractères spéciaux ($2Ø - $7F)

Éventuellement, 128 caractères définis par l'utilisateur ($8Ø $FF)

Les caractères de contrôle (CO)



Les caractères CHR$(Ø) à CHR$(31) sont un petit peu abstraits. Il ne s'agit plus exactement d'une action d'écriture mais d'une action, par exemple, de déplacement du curseur ou même de l'arrêt du fonctionnement d'un programme. Le tableau CO ci-contre donne la signification de ces caractères. Ainsi, plutôt que d'écrire CHR$(12) qui, comme la touche RAZ, efface l'écran, il est possible de frapper ce caractère au clavier grâce à la touche CNT (contrôle). Précisément CNT L, noté souvent Lc, est une frappe équivalente à PRINT CHR$(12). Les caractères de contrôle Ac à Zc permettent ainsi de simuler les actions indiquées dans le tableau CO. Quel plaisir de frapper CNT J et de voir le curseur descendre comme si l'on avait frappé sur la touche ↓ ! On peut considérer que la pression de la touche CNT met en correspondance les cases des colonnes 64 et 8Ø de GØ avec les cases de CO; malheureusement, quatre des cinq dernières cases de cette grille ne correspondent à aucune touche du clavier. Il n'y a donc aucun moyen de les frapper directement. Il en est ainsi du caractère CHR$(27), plus explicitement (SIC), appelé « ESCAPE ».

Lecture du tableau CO :

La case Kc indique l'équivalence de quatre « caractères » :

- le caractère de code ASCII « onze » : CHR$ (11)
- le caractère de contrôle Kc : CNT K
- la touche du clavier TO7 : (flèche en haut)
- le mnémonique anglophone de tabulation verticale : VT

Le tableau ci-dessous indique la signification des mnémoniques anglophones des « caractères de contrôle ».

BEL	(Bell)	Sonnerie
BS	(Backspace)	Retour arrière
CAN	(Cancel)	Annulation
CR	(Carriage return)	Retour de chariot
ENQ	(Enquiry)	Demande
EOT	(End of transmission)	Fin de communication
ESC	(Escape)	Échappement
ETX	(End of text)	Fin de texte
FF	(Form feed)	Présentation de formule
HT	(Horizontal tabulation)	Tabulation horizontale
LF	(Line feed)	Interligne
NUL	(Null)	Nul
REP	(Repeat)	Répétition
RS	(Record separator)	Réparateur d'article
SI	(Shift-in)	En code) semi-
SO	(Shift-out)	Hors code) graphiques
SOH	(Start of heading)	Début d'en-tête
SP	(Space)	Espace
STX	(Start of text)	Début de texte
US	(Unit separator)	Séparateur de sous-article
VT	(Vertical tabulation)	Tabulation verticale

Remarques

— Accès à la police de caractères vidéotex G1 : CHR$ (14) ou Nc.
— Sortie de la police de caractères vidéotex G1 : CHR$ (15) ou Oc.
— Accès à la police de caractères G2 : CHR$ (22) ou Vc ou ACC.
— L'accès à ESCAPE par CHR$ (27) n'est pas directement possible sur le clavier du TO7.

— DC2/REP/Rc. La séquence (Rc/X) transmise à PUTCH permet de répéter *n* fois le caractère précédent le Rc. *n* est le nombre obtenu en considérant les six bits de poids faible de X, dont le code est supposé compris entre $40 et $7F. PRINT « A » CHR$ (18) « e » produira l'affichage de 38 fois la lettre A. Notez que les « ; » ne sont pas indispensables dans la syntaxe de PRINT. Notez également que le code ASCII de « e » est $65 = %Ø11ØØ1Ø1. Les six bits de poids faible indiquent 1ØØ1Ø1 = 37 répétitions. En assembleur :

```
                0012    REP     EQU     $12


5000                            ORG     $B000

B000 C6    41                   LDB     #'A
B002 BD    E803                 JSR     $E803
B005 C6    12                   LDB     #REP
B007 BD    E803                 JSR     $E803
B00A C6    65                   LDB     #37.OR.64
B00C 7E    E803                 JMP     $E803

                0000            END

00000 Total Errors
```

— CR/Mc. Remarquez que l'envoi d'un CR à PUTCH ne produit pas de LF (« line feed » ou saut de ligne).

L'escapage

Les « séquences ESCAPE » permettent de réaliser beaucoup de choses. Il existe deux types de « séquences ESCAPE », les séquences permettant d'initialiser un attribut courant (ESC/X) et les séquences permettant de donner un attribut plein écran (ESC/#/SP/X, où SP représente l'espace, CHR$(32)). La différence entre ces deux séquences rappelle celle qui existe entre COLOR (attribut courant) et SCREEN (attribut plein écran).

Codes reconnus par les séquences ESCAPE

Grille G1	MSB	4	5	6
LSB		caractères	fonds	cadre
Ø		noirs	noirs	noir
1		rouges	rouges	rouge
2		verts	verts	vert
3		jaunes	jaunes	jaune
4		bleus	bleus	bleu
5		magentas	magentas	magenta
6		cyans	cyans	cyan
7		blancs	blancs	blanc
8			masquage	écriture forme
9				écr.forme et couleur
A				défilement normal
B				mode page
C		taille normale	vidéo inverse	mémoire couleur Ø
D		double hauteur		mémoire couleur 1
E		double largeur		défilement doux
F		double taille	démasquage	

Sur le TO7-7Ø, on obtient les demi-teintes en forçant le bit de poids fort à 1.

La commande US

Cette commande permet de définir la fenêtre de texte (comme on le ferait à l'aide des deux premiers arguments de CONSOLE). Elle permet également de placer le curseur en une position déterminée de l'écran (comme on le ferait à l'aide des deux premiers arguments de LOCATE). La syntaxe est (US/Y/X). L'interprétation de la séquence dépend des bits de poids fort de Y et de X.

Si Y et X sont dans la zone $10 - $19, la séquence détermine la ligne du bas de la fenêtre de texte, LIGNE = 10 * n + n', où n et n' sont les nombres obtenus en considérant les quatre bits de poids faible de Y et X.

Si Y et X sont dans la zone $2Ø - $29, la séquence détermine la ligne du haut de la fenêtre de texte. Le calcul est mené comme ci-dessus.

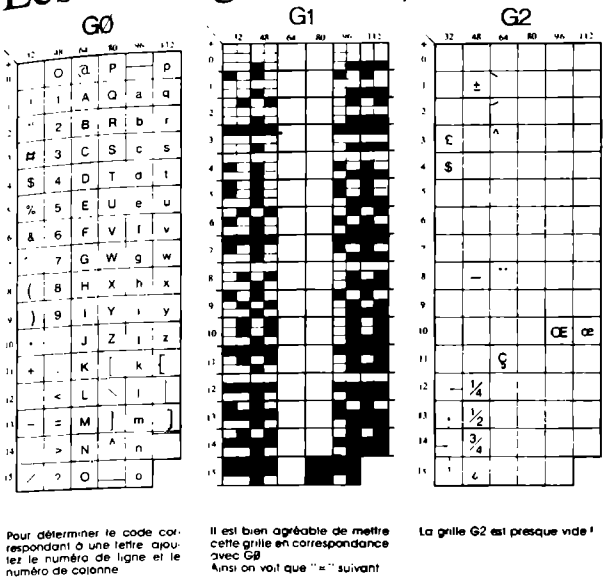
Si Y et X sont dans la zone $3Ø -$39, la séquence place le curseur au début de la ligne dont le numéro est calculé comme ci-dessus.

Si Y et X sont dans la zone $4Ø - $7F, le curseur est placé sur la n^{ieme} ligne et la n^{ieme} colonne. n est le nombre obtenu en considérant les six bits de poids faible de Y. n' est le nombre obtenu en considérant les six bits de poids faible de X.

```
                E803    PUTC    EQU     $E803
                001F    US      EQU     $1F

B800                            ORG     $B800

B800 34   04                    PSHS    B
B802 C6   1F                    LDB     #US
B804 8D   1C                    BSR     PTC

                        * CHIFFRE DES UNITES

B806 E6   E4                    LDB     ,S
B808 C1   0A            BI0     CMPB    #10
B80A 2B   04                    BMI     SI0
B80C C0   0A                    SUBB    #10
B80E 20   F8                    BRA     BI0
B810 8D   0C            SI0     BSR     SI1

                        * CHIFFRE DES DIZAINES

B812 5F                         CLRB
B813 35   02                    PULS    A
B815 81   0A            BI1     CMPA    #10
B817 2B   05                    BMI     SI1
B819 5C                         INCB
B81A 80   0A                    SUBA    #10
B81C 20   F7                    BRA     BI1

                        * ENVOI A PUTC

B81E C4   0F            SI1     ANDB    #$0F
B820 CA   20                    ORB     #$20
B822 7E   E803          PTC     JMP     PUTC

                0000            END

00000 Total Errors
```

Si B contient un nombre entre Ø et 24, ce programme initialise le haut de la fenêtre texte.

Les trois grilles GØ, G1, G2



GØ

La police de caractères habituelle est représentée sur la grille GØ de manière à faire apparaître le code ASCII de chaque caractère. Vous savez certainement que l'on peut imprimer la lettre A en écrivant, au lieu de PRINT « A », l'instruction PRINT CHR$ (65). D'une manière plus générale, si le caractère « G » a pour code ASCII « g », alors CHR$ (g) est un autre nom de « G ».

Voici, par exemple, un programme qui transforme les minuscules en majuscules :

```
10 INPUT R$
20 IF LEN(R$)=0 THEN 100
25 FOR I=1 TO LEN(R$)
30 L=ASC(MID$(R$,I,1))
40 IF L>96 AND L<122 THEN MID$(R$,I,1)=C
HR$(L-32)
50 NEXT I
60 ' Réponse
70 ....
80 ....
90 ....
100 ' Réponse vide
101 ....
102 ....
```

G1

Après avoir envoyé le code SO (CHR$(14)), PUTCH interprète et affiche les caractères semi-graphiques jusqu'à la réception du code SI (CHR$(15)) qui rétablit l'affichage des caractères de GØ.

Les caractères de G1 sont constitués à partir de six petits pavés dont l'état allumé ou éteint, dépend de l'état des bits de l'octet définissant le caractère.

Ø	b6	1	b4	b3	b2	b1	b0

b0	b1
b2	b3
b4	b6

```
10 CLS:SCREEN 3,4,4
20 CO=&H20
30 LOCATE 0,0
35 PRINT CHR$(15)"CODE $"HEX$(CO)" "CHR$
(14)CHR$(CO)
40 A$=INKEY$:IF A$="" THEN 40
50 IF A$="A" THEN CO=CO XOR 1
51 IF A$="Z" THEN CO=CO XOR 2
52 IF A$="Q" THEN CO=CO XOR 4
53 IF A$="S" THEN CO=CO XOR 8
54 IF A$="W" THEN CO=CO XOR 16
55 IF A$="X" THEN CO=CO XOR 64
60 GOTO 30
OK
```

Voici un petit éditeur qui vous permet de composer les caractères vidéotex à l'aide des touches A,Z,Q,S,W et X du clavier. Pour bien comprendre les instructions 5Ø à 55, il faut écrire les valeurs 1,2,4,8,16 et 64 en binaire et regarder de près l'opération XOR (OU exclusif).

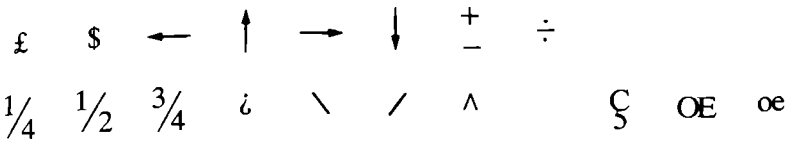
```
 1 : 00000001
 2 : 00000010
 4 : 00000100
 8 : 00001000
16 : 00010000
64 : 01000000
```

X	Y	X XOR Y
1	1	Ø
1	Ø	1
Ø	1	1
Ø	Ø	Ø

On constate que 1 XOR Y = $\overline{Y}$ (Y est inversé)
et que Ø XOR Y = Y (pas de changement).

G2

Il existe une troisième grille de caractères qu'il est possible d'utiliser l'espace d'un instant. Le caractère de contrôle SS2, ou encore Vc obtenu au clavier grâce à la touche ACC, ouvre l'accès à la grille G2 le temps d'un caractère. Le retour à GØ est automatique. Les caractères affichables dans G2 sont les suivants :



C'est ainsi que les minuscules accentuées sont codées sur trois octets : le code ACC ($16), le code de l'accent, et enfin le code de la lettre à accentuer. Par exemple, $16, $41, $65 codent le « è », et le mot « l'œuvre » s'écrira PRINT « l' »CHR$(22)« zuvre », ce qui ne lasse pas de me plaire.

Un programme d'illustration

Puisque CHR$(27) ne correspond à aucune touche du TO7, on peut le « simuler » en utilisant un caractère de contrôle inutilisé, par exemple CNT U = CHR$(21). De même, pour avoir accès aux caractères auxquels ne correspond pas de touche, nous passons par CNT Y. Le caractère suivant est alors « masqué » et les caractères des colonnes 32, 64 et 112 sont compris comme ceux des colonnes 48, 8Ø et 128. On peut ainsi obtenir la vidéo inverse par :

Uc Yc Mxxx...

```
1  '
2  '       Sous les claviers, les pages
3  '
10 B$=INKEY$:IF B$="" THEN 10
20 CO=ASC(B$)
30 IF CO=21 THEN CO=27:'Ctrl-U pour ESC
35 IF CO=25 THEN 100  :'Ctrl-Y pour MOD.

38 PRINT CHR$(CO);
40 GOTO 10
100 '
101 '    Transformation du clavier
102 '
```

```
103 '    K devient [   k devient {
104 '    L devient \   l devient |
105 '    M devient ]   m devient }
106 '    N devient ↑   n devient ~
107 '    O devient _   o devient ■
108 '
115 B$=INKEY$:IF B$="" THEN 115
120 CC=ASC(B$)+16
125 GOTO 35
```

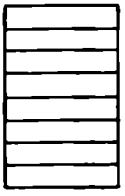
Ce petit programme vous permet d'étudier les trucs décrits ci-dessus; tout le clavier est à présent plein de sous-entendus. UcO, et les caractères grossissent... ARcZ, et voici plein de A... YcK devient un crochet... Nc, et vous voilà parti dans le semi-graphique. Pour quelqu'un de non averti qui vous regarde frapper sur le clavier, vous faites alors figure d'un vrai sorcier : vous frappez E, et U s'affiche ! Vous frappez I, et c'est Y qui apparaît !

Diable ! L'informatique est complètement démystifiée.

L'intérêt principal de tout ceci réside dans le fait que l'on peut décrire entièrement une page-écran TO7 dans une chaîne de caractères très compacte, PUTCH permettant de l'afficher très rapidement. C'est ainsi qu'est définie en ROM la page d'en-tête qui apparaît à l'initialisation du système.

Caractères définis par l'utilisateur

Tous les caractères dont les codes sont compris entre $8$0 et $FF sont considérés comme des caractères utilisateurs. Chaque caractère est défini par huit octets de RAM. Si BASIC gère cette table RAM au moyen des instructions CLEAR (troisième paramètre), DEFGR$ et GR$, il peut être bon de savoir qu'il existe un pointeur qui permet de connaître le début de la table des caractères définissables. Ce pointeur seize bits se nomme USERAF et il se trouve aux adresses $6Ø2D - $6Ø2E. Il donne l'adresse de l'octet du bas du caractère de code $8Ø. On trouve ensuite les sept octets qui définissent le caractère.



← USERAF

Le caractère GR$(n) a pour code $80 + n. Il est défini par les octets situés entre USERAF + 8 * n et USERAF + 8 * (n+1) -1.

Bien entendu, ceci permet de créer plusieurs tables de caractères définissables et de les utiliser tout à loisir. On commence d'ailleurs à voir circuler des programmes TO7 qui écrivent en caractères grecs et en minuscules liées... On peut ainsi avoir facilement en magasin de nombreuses polices de caractères.

Loupes et écriture penchée

Le TO7 a naturellement quelque part dans sa mémoire (ROM) une zone qui contient les octets qui permettent de définir les caractères de GØ et de G2 de la même manière que sont définis les caractères utilisateurs (les caractères de G1 sont générés par programme). Cette zone commence en &HE845 par huit octets à zéro qui correspondent à l'espace (ASCII 32). On trouve ensuite les autres caractères de l'alphabet GØ.

Le contenu des huit octets définissant le caractère C$ est donné par :

```
5 INPUT"Caractere :";C$
10 C=ASC(C$)-32
15 FOR I=7 TO 0 STEP-1
20 D=PEEK(&HE845 + 8*C + I)
25 PRINT HEX$(D)
30 NEXT I
```

Application 1 : pour les très très myopes

On peut facilement remplacer la grille de 8 x 8 points d'écran par un rectangle 8 x 8 caractères.

```
10 SCREEN 3,4,4:CLS
15 C$="@":X=5:Y=12
20 C=ASC(C$)-32
25 FOR I=0 TO 7
30 D=PEEK(&HE845 + 8*C + I)
31 LOCATE 0,Y-I:PRINT HEX$(D)
35 FOR B=0 TO 7
40 M$="*"
45 IF (D AND (2^B))=0 THEN M$="
50 LOCATE X+7-B,Y-I
```

```
55 PRINT M$;
60 NEXT B
65 NEXT I
70 LOCATE 0,0:END
```

Commentaires : ligne 15 - il est évident que ceci peut être modifié pour une autre lettre ou même un INPUT ou encore un MID$ de quelque chose.
Ligne 20 - les trente-deux premiers caractères ASCII ne sont pas affichables (ce sont les codes de contrôle).
Ligne 45 : il faut penser binaire ! La condition logique est vérifiée si le bit B est à 1.

Application 2 : pour les désaxés

Si on connaît un peu l'organisation de la page graphique, on peut afficher les caractères presque n'importe où.

```
AØ
5  SCREEN 3,4,4:CLS
10 POKE &HE7C3,PEEK(&HE7C3) OR 1
15 M$="Bonjour"
20 M=&H5000
25 FOR L=1 TO LEN(M$)
30 C=ASC(MID$(M$,L,1))-32
35 FOR I=0 TO 7
40 D=PEEK(&HE845 + 8*C + I)
45 POKE M-40*I,D
50 NEXT I
55 M=M-39:NEXT L
```

Commentaires : ligne 10 - la zone mémoire correspondant à l'écran va de &H4ØØØ jusqu'à &H5F3F. Selon que le bit Ø de &HE7C3 est à 1 ou Ø, on travaille dans la mémoire « formes » ou dans la mémoire « couleurs ». La ligne 1Ø force le bit Ø de &HE7C3 à 1.

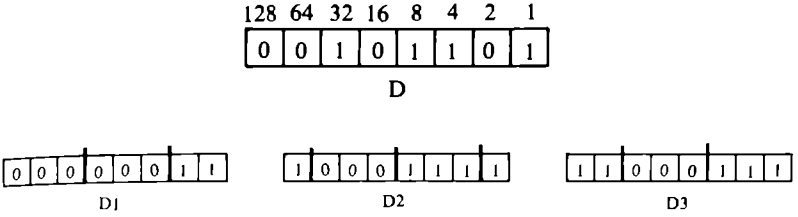
Ligne 20 - ceci fixe l'origine des écritures. Il y a deux cents lignes de quarante colonnes d'un octet. Le premier octet en haut à gauche a pour adresse &H4ØØØ. Si L (Ø<L<2ØØ) est le numéro de ligne et C (Ø<C<4Ø) le numéro de colonne de l'octet où l'on souhaite écrire, son adresse est &H4ØØØ + 4Ø * L + C (BASIC supporte très bien cette arithmétique composite décimale/hexadécimale).

Lignes 45-55 - diminuer l'adresse mémoire de 4Ø revient à écrire dans l'octet immédiatement au-dessus du précédent. Augmenter

l'adresse mémoire de 1 revient à écrire dans l'octet immédiatement à droite du précédent.

Application 3 : pour les pas trop myopes

On sait, grâce à l'instruction ATTRB 1,1, afficher les caractères en double taille. Je propose un programme permettant de les afficher en triple taille.



Ecrire en triple taille suppose de créer trois octets, D1, D2, et D3, en « étirant » les octets de définition D de la manière indiquée dans le schéma ci-dessus.

```
5  SCRLN 3,4,4:CLS
10 POKE &HE7C3,PEEK(&HE7C3) OR 1
15 M$="Bonjour"
20 M=&H5000 - 10
25 FOR L=1 TO LEN(M$)
30 C=ASC(MID$(M$,L,1))-32
35 FOR I=0 TO 7
40 D=PEEK(&HE845 + 8*C + I)
45 D1=224*(D AND 128)/128+28*(D AND 64)/
64+3*(D AND 32)/32
50 D2=128*(D AND 32)/32+112*(D AND 16)/1
6+14*(D AND 8)/8+(D AND 4)/4
55 D3=192*(D AND 4)/4+56*(D AND 2)/2+7*(
D AND 1)
60 FOR E=0 TO 2
65 POKE M-40*(3*I+E),D1
66 POKE M-40*(3*I+E)+1,D2
67 POKE M-40*(3*I+E)+2,D3
70 NEXT E
80 NEXT I
85 M=M+3:NEXT L
```

Commentaires : tout ceci est beaucoup trop compliqué ! Toutefois, si vous êtes, comme moi, totalement masochiste, je peux vous dire

que dans les lignes 45, 5Ø et 55, les quantités du style (D AND 64)/64 valent Ø ou 1 selon que le bit correspondant de D est à Ø ou à 1. (Il y a peut-être plus simple, mais je suis matheux... !) Les facteurs multiplicatifs de ces quantités correspondent aux diverses configurations de trois bits (2 ou 1 aux extrémités) dans les octets D1, D2 et D3. On peut facilement réduire la hauteur des lettres (ou l'augmenter) en lignes 6Ø à 7Ø.

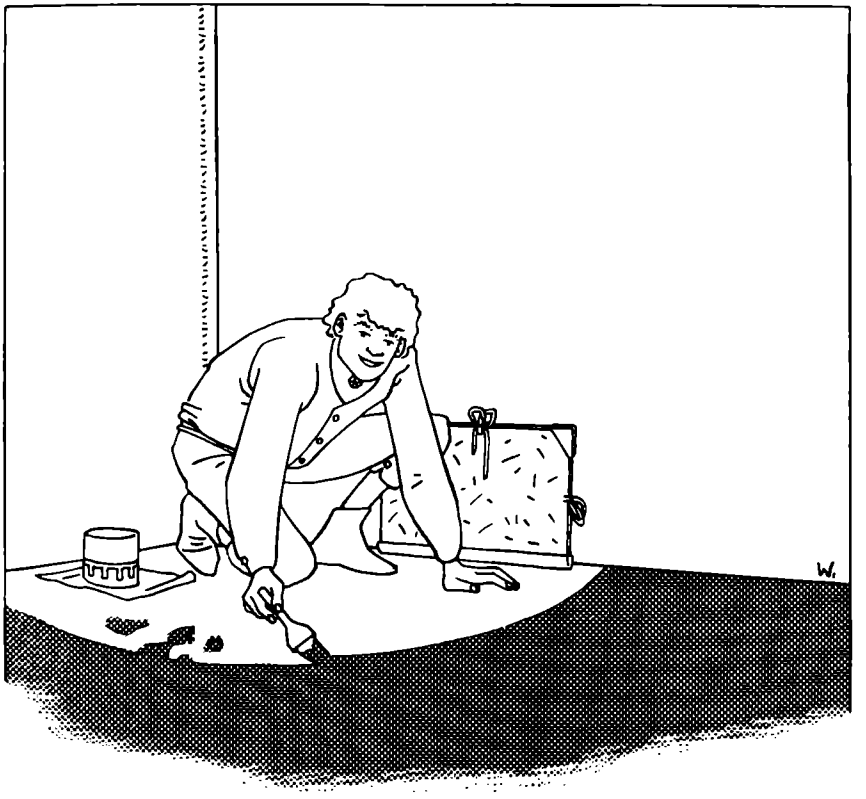
Pour terminer j'invite le lecteur à concocter un programme « quadruple taille » qui sera plus simple, ou « heptuple taille », sensiblement plus complexe.

Je signale qu'il est parfaitement possible d'écrire à cheval sur deux octets. Il suffit de travailler le décalage. Curieusement, ce type de programme s'écrit relativement plus simplement en assembleur qu'en BASIC, ce qui présente en plus l'avantage d'être exécuté nettement plus rapidement. Signalons pour finir que si l'on place le pointeur des caractères utilisateurs (USERAF) sur le début des caractères connus par le TO7, GR$(Ø) devient le blanc, GR$(1) devient « ! », etc. On trouvera dans les derniers GR$ ainsi découverts les caractères G2 et les éléments qui servent à réaliser la page d'initialisation.

```
POKE &H602D,&HE8
POKE &H602E,&H4D

PRINT GR$(5)
```

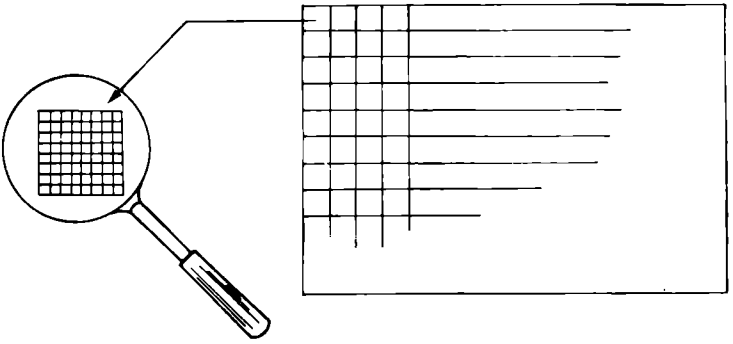
etc.

Curseur et page graphique



La mémoire d'écran

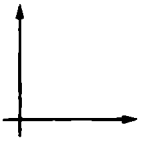
Tout ce qui est affiché sur l'écran de télévision du TO7 est l'image de certaines configurations de 0 et de 1 dans la mémoire du TO7. Vous savez sans doute que le TO7 permet d'afficher vingt-cinq rangées[1] de quarante caractères numérotés respectivement de Ø à 24 et de Ø à 39, chaque caractère étant lui-même constitué de 8 x 8 petits points élémentaires ou « pixels ».



La mémoire d'écran du TO7 contient les informations relatives à ces (25 x 8) x (40 x 8) = 200 x 320 = 64 000 points.

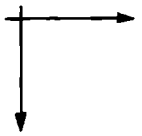
Ces points sont repérés par leurs coordonnées, l'abscisse **X** (Ø<X<319) et l'ordonnée Y (Ø<Y<199).

Il faut remarquer que le système de coordonnées de la machine n'est pas le système habituellement utilisé par les mathématiciens qui préfèrent généralement placer l'origine dans le coin inférieur gauche et graduer les ordonnées dans le sens croissant vers le haut.



Ici, l'origine est dans le coin supérieur gauche et si la graduation des abscisses va bien de la gauche vers la droite, celles des ordonnées va vers le bas !

1. Le terme « rangée » est utilisé à la place du terme « ligne » quand on souhaite éviter la confusion entre une « ligne de télévision » (il y a 625 sur une image télévision normale) et une « ligne de texte » (il y en a vingt-cinq !).



En fait, c'est très naturel car c'est bien ainsi que le texte est lu, du haut en bas. Or, l'écran est avant tout fait pour travailler du texte.

La zone mémoire réservée à l'écran est située entre les adresses &H4ØØØ et &H5FFF. Cette zone d'adresses comporte donc $2ØØØ = 8192 = 8 K octets. Or, 64 000 points affichés tiennent, à raison de huit points par octet, dans 8 000 octets. Il y a donc 192 petits octets qui ne servent à rien. Les points affichés sur l'écran correspondent aux bits des octets d'adresses &H4ØØØ à &H5F3F. Les octets d'adresses &H5F4Ø à &H5FFF sont disponibles pour l'utilisateur. Ils sont néanmoins rarement utilisés car assez délicats à atteindre.

Un point peut être allumé ou éteint. Il y a 64 000 points sur l'écran du TO7 ; il est donc possible de dessiner 2^{64000} (2 puissance 64 000) dessins différents sur l'écran du TO7. (2^{64000} vaut à peu près 10^{19200}, c'est-à-dire 1 avec 19 200 zéros derrière. Il faudrait déjà un bon fascicule pour écrire ce nombre. En cherchant à l'apprécier, on peut s'amuser à compter combien de temps il faudrait pour les passer tous en revue à raison, mettons, d'un million à la seconde. Même en comptant en nombre d'univers, sachant que le nôtre date d'il y a environ 15 milliards d'années, on constate qu'il est totalement impossible d'imaginer ce nombre !)

Et encore... ce calcul de combinatoire ne tient pas compte de la couleur, car le TO7 sait afficher de la couleur, précisément huit couleurs pour le TO7 et seize pour le TO7/70.

Malheureusement, les huit points d'un même octet ne peuvent apparaître qu'en deux couleurs. Les bits à 1 seront d'une couleur et les bits à Ø d'une autre couleur. On parle de la couleur POINT pour les bits à 1 et de la couleur FOND pour les bits à Ø.

Interlude

Voici un portrait de Huygens, obtenu par une chance assez incroyable à l'aide du petit programme ci-dessous qui génère des dessins aléatoires.

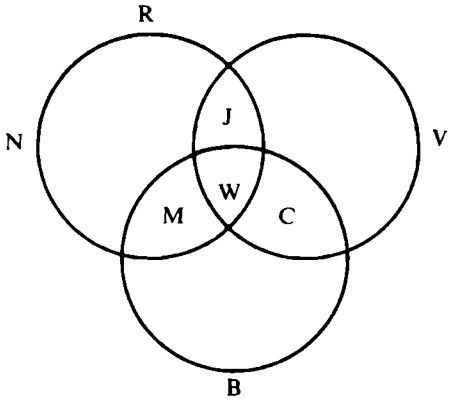
```
10 CLS
16 LOCATE 0,0,0
20 A$=INKEY$
25 PRINT "     Dessin aléatoire
30 I=RND:A$=INKEY$:IF A$="" THEN 30
40 FOR M=&H4000 TO &H5F3F
50 D=INT(256*RND)
60 POKE M,D
70 NEXT M
80 GOTO 40
OK
```

Les couleurs

Le code des couleurs n'est pas le fait du hasard.

code	B	V	R	couleur	
ø	ø	ø	ø	noir	N
1	ø	ø	1	rouge	R
2	ø	1	ø	vert	V
3	ø	1	1	jaune	J
4	1	ø	ø	bleu	B
5	1	ø	1	magenta	M
6	1	1	ø	cyan	C
7	1	1	1	blanc	W

La synthèse de la couleur sur un écran de télévision est réalisée à partir de l'addition de trois couleurs fondamentales : le rouge, le vert et le bleu.

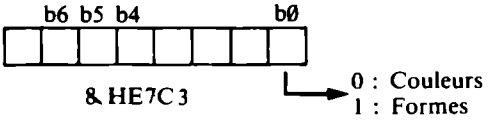


On peut imaginer trois projecteurs qui envoient des taches de lumière sur un mur blanc. Là où aucune lumière n'arrive, on voit

du noir. Là où les différents faisceaux se croisent, les couleurs s'ajoutent, par exemple, de la lumière verte ajoutée à de la lumière rouge donne de la lumière jaune. Le codage binaire des couleurs devient alors très clair, les codes choisis reflètant simplement la synthèse additive de la lumière.

Attention, les imprimeurs et les peintres travaillent en synthèse soustractive des couleurs : plus on ajoute de teinte, plus celle-ci tire vers le noir. Leurs fondamentales sont les complémentaires, jaune, magenta et cyan.

Les informations concernant la colorimétrie d'un octet donné sont elles-mêmes stockées dans... un octet ! De manière très habile, les octets correspondant à la forme du dessin et les octets correspondant à la couleur du dessin résident à la même adresse, entre &H4ØØØ et &H5F3F. Il existe dans le TO7 un commutateur qui permet à l'utilisateur de travailler au choix dans la mémoire des formes ou dans la mémoire des couleurs. Ce commutateur est le bit bØ de l'octet &HE7C3. Quand celui-ci est à 1, on travaille dans la mémoire des formes, et quand celui-ci est à Ø on travaille dans la mémoire des couleurs.



Essayez donc les manipulations suivantes :

Remise à zéro

POKE &HE7C3,1 : POKE &H4ØØØ,25.

Outre le fait que la bordure de l'écran du TO7 devient noire, ce qui est dû à la mise à Ø de b4, b5 et b6, vous verrez apparaître quelques points au-dessus du P de POKE. Ces points correspondent à la configuration binaire de 25 :

0	0	0	1	1	0	0	1

Tout ceci explique que, pour sauvegarder ou pour restituer une page écran avec ses couleurs, il faut manipuler deux fichiers de 8K.

```
POKE &HE7C3,1:SAVEM "F",&H4000,&H5F40,0

POKE &HE7C3,0:SAVEM "C",&H4000,&H5F40,0

POKE &HE7C3,1:LOADM "F"
POKE &HE7C3,0:SAVEM "C"
```

Attention, quand on écrit sur une ligne :

```
POKE &HE7C3,0
OK
```

il ne faut pas imaginer que le système laisse le bØ de E7C3 à Ø. Le simple fait d'écrire le OK implique un travail dans la mémoire graphique. Si l'on tape à la ligne suivante :

```
PRINT PEEK(&HE7C3)
 1
OK
```

on aura parfois la surprise d'obtenir 1 pour résultat. En fait, s'il n'y a pas de défilement d'écran, on obtiendra 1, s'il y a un défilement, on obtiendra Ø. C'est pour cette raison que les instructions en mode immédiat données dans ce chapitre sont écrites sur une seule ligne en utilisant les « : ».

Si on désire être moins brutal avec &HE7C3 et ne modifier que bØ sans toucher aux sept autres bits, on peut écrire :

```
POKE &HE7C3,PEEK(&HE7C3) OR 1
```

pour travailler la couleur et

```
POKE &HE7C3,PEEK(&HE7C3) AND 254
```

pour travailler les formes.

D'ailleurs les sept bits de poids fort n'ont pas la même signification sur TO7 et TO7/70. Il est prudent de les laisser en paix !

Pour comprendre ces instructions, il faut écrire les valeurs 254 et 1 en binaire et regarder sur les tables le résultat des opérations AND et OR.

```
254: 11111110
  1: 00000001
```

X	Y	X AND Y	X OR Y
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

Pour AND, on constate que 1 AND Y = Y (pas de changement) et que Ø AND Y = Ø (forcé à Ø).

Pour OR, on constate que Ø OR Y = Y (pas de changement) et que 1 OR Y = 1 (forcé à 1).

Dans les instructions BASIC précédentes, le 254 et le 1 sont des masques qui servent à forcer certains bits à Ø ou à 1.

Il est possible d'utiliser deux masques pour réaliser en deux opérations les quatre opérations suivantes :

Données	masque	masque	Résultats :
M	X	Y	(M AND X) XOR Y
n	1	1	n (complémentation)
n	1	Ø	n (pas de modification)
n	Ø	1	1 (forcé à 1)
n	Ø	Ø	Ø (forcé à Ø)

Suite de manipulations

En supposant que vous ayez laissé dans l'octet d'adresse &HØØØØ une certaine configuration de 0 et de 1, vous pouvez jouer à présent avec les couleurs en essayant :

```
POKE &HE7C3,Ø:POKE &HØØØØ,XX
```

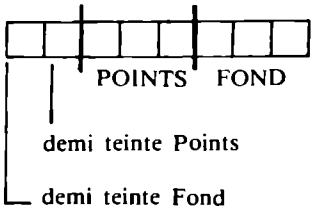
où XX est une valeur comprise entre Ø et 255, plus précisément, d'ailleurs, entre Ø et 63 sur le TO7. La colorimétrie de l'octet se calcule ainsi : XX = 8 * P + F, où Ø<P≤7 est la couleur des points et Ø<F≤7 est la couleur des fonds.

Sur TO7, les deux bits de poids fort ne servent à rien. En outre, le système les force systématiquement à 1. Bien que la mémoire couleur soit en théorie une mémoire de 8 K RAM, en fait, deux bits de chacun des 8 192 octets de cette page sont totalement inutilisables. L'expérience suivante devrait vous en convaincre.

```
POKE &HE7C3,Ø:POKE &HØØØØ,Ø:PRINT HEX$(&
HØØØØ)
```

vous renvoie 192, qui s'écrit 11ØØØØØØ en binaire. Seuls les six bits de poids faible des mémoires couleur sont utilisables.

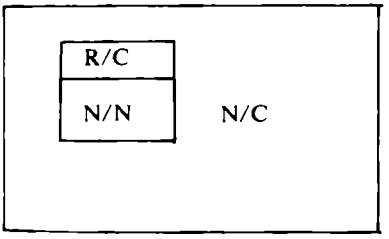
Sur le TO7/70, les deux bits de poids fort servent à créer les demi-teintes. On peut donc utiliser n'importe quelle valeur entre Ø et 255 pour XX, sachant que l'octet couleur a la structure suivante :



L'effet caméléon

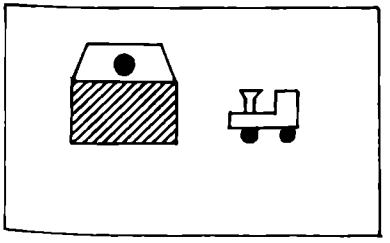
Le travail sur les formes et sur les couleurs est donc totalement indépendant. D'ailleurs, en BASIC, l'instruction SCREEN travaille uniquement sur la couleur sans affecter la forme et la mise à 1 du troisième paramètre de CONSOLE qui permet de travailler sur les formes sans modifier la couleur. C'est ainsi que le petit train de J. Delcourt rentre en gare : le train est dessiné grâce à des GR$, la couleur a été initialisée grâce à des BOXF avec des paramètres de couleur positifs (couleurs des points) et négatifs (couleurs des fonds).

On exécute ensuite un CONSOLE,,1 qui bloque la mise à jour de la mémoire des couleurs.



page couleur

	points	fonds
N/C	NOIR	CYAN
N/N	NOIR	NOIR
R/C	ROUGE	CYAN



Jean déplace alors son train par un jeu d'affichage de GR$ et d'espaces. Quand le train est affiché sur le mur de la gare en N/N; on ne le voit plus. On a l'impression qu'il passe derrière.

On peut également réaliser sur ce principe des génériques sympathiques :

```
10 A$="BONJOUR"
20 SCREEN 4,4,4
30 BOXF(0,11)-(39,11),,,6
31 BOXF(0,12)-(39,12),,,5
32 BOXF(0,13)-(39,13),,,4
33 BOXF(0,14)-(39,14),,,3
34 BOXF(0,15)-(39,15),,,2
40 CONSOLE ,,1,1
50 FOR I=0 TO 30
55 LOCATE 10,24
60 IF I<10 THEN PRINT A$
61 IF I>9 THEN PRINT
65 NEXT I
70 CONSOLE ,,0,0
OK
```

Le quatrième paramètre de CONSOLE en ligne 4Ø sert à initialiser le défilement doux.

Le curseur

Le curseur indique l'endroit où le prochain PRINT sera effectué. Il est possible de placer le curseur où l'on veut grâce à l'instruction LOCATE, ou grâce à une séquence assembleur US/Y/X. Il est également possible d'afficher le curseur clignotant ou de le faire disparaître grâce au troisième paramètre de LOCATE ou à l'aide de l'interprétation d'un DC1 ou d'un DC4.

Le curseur est un octet qui clignote de façon régulière. Le système génère une interruption pour complémenter l'état des huit bits de l'octet correspondant à l'adresse où se trouve le curseur. La position du curseur est contenue dans le pointeur seize bits SCRPT ($6Ø5A - $6Ø5B). Il est possible d'afficher des caractères un peu n'importe où sur l'écran en modifiant les valeurs contenues à cette adresse :

```
10 CLS
15 LOCATE 15,6,0
20 A$="BONJOUR"
30 FOR I=1 TO LEN(A$)
40 PRINT MID$(A$,I,1);
50 SCRPT=256*PEEK(&H605A)+PEEK(&H605B)
60 SCRPT=SCRPT+40
70 POKE &H605A,SCRPT @ 256
```

```
80 POKE &H605B,SCRPT MOD 256
90 NEXT I
```

```
BONJOUR
```

Pour calculer l'adresse physique du curseur, il faut se souvenir que la page graphique commence en $4000 et qu'il y a quarante octets par ligne. Un décalage d'un octet correspond à un décalage horizontal d'un caractère. Un décalage de quarante octets correspond à un décalage vertical d'une ligne. Un décalage de 320 octets (8 x 40) correspond à un décalage vertical d'un caractère.

Bidouilles graphiques

Compter

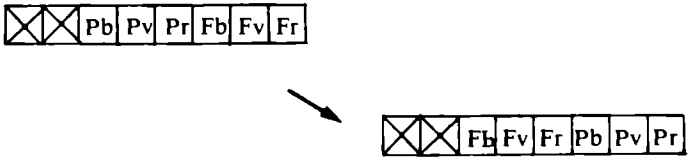
J'ai un petit faible pour :

```
PRC     EQU     $E7C3
PUTC    EQU     $E803

        ORG     $B800

CLS     LDB     $12
        JSR     PUTC

FORME   LDB     PRC
        ORB     #1
        STB     PRC

B0      INC     $5000
        BNE     B0
        INC     $5001
        BNE     B0
        RTS

       .END
```

Ce petit programme compte sous vos yeux de 0 à 65 536 en environ une seconde. Essayez donc de voir ce que donne :
FOR I = 0 TO 65 536
NEXT I
au point de vue du temps !

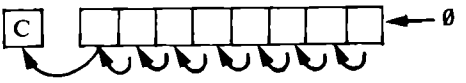
Commutation des pages couleurs

Voici un petit programme assembleur qui chamboule les couleurs : les couleurs de fond et de points sont échangées sur tout l'écran. Chaque octet couleur est ainsi trafiqué :

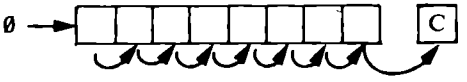



```
            E7C3    PRC     EQU     $E7C3
            4000    PAGDEB  EQU     $4000
            5F40    PAGFIN  EQU     $5F40

BF00                        ORG     $BF00

                    * Travail sur les coul
BF00 B6     E7C3            LDA     PRC
BF03 84     FE              ANDA    #$FE
BF05 B7     E7C3            STA     PRC

BF08 8E     4000            LDX     #PAGDEB
BF0B A6     84      B0      LDA     ,X
BF0D 84     07              ANDA    #7
BF0F 48                     LSLA
BF10 48                     LSLA
BF11 48                     LSLA
BF12 34     02              PSHS    A
BF14 A6     84              LDA     ,X
BF16 84     38              ANDA    #$38
BF18 44                     LSRA
BF19 44                     LSRA
BF1A 44                     LSRA
BF1B AA     E0              ORA     ,S+
BF1D A7     80              STA     ,X+
BF1F 8C     5F40            CMPX    #PAGFIN
BF22 26     E7              BNE     B0
BF24 39                     RTS

            0000            END

00000 Total Errors
```

LSL : LOGICAL SHIFT LEFT
L'octet est décalé d'un rang vers la gauche.



LSR : LOGICAL SHIFT RIGHT
La même chose mais vers la droite !



$7 = Ø Ø Ø Ø Ø 1 1 1

Masque « fond »

$38 = Ø Ø 1 1 1 Ø Ø Ø

Masque « points »

Des bazars

Je propose, dans le programme suivant, de dessiner sur l'écran un motif (grâce, par exemple, au BASIC), puis d'appeler un programme assembleur qui pratique un « OU exclusif » entre chaque octet de dessin et un octet du même dessin lu plus loin dans la mémoire d'image. Ceci revient au même que si l'on réalisait un « OU exclusif » entre le dessin et lui-même translaté. Le programme boucle : la seule manière de l'arrêter est de faire un RESET. Les valeurs données pour le décalage doivent être comprises entre 1 et 8ØØØ (&H1F4Ø). Un décalage de 1 donnera plutôt un effet de décalage horizontal. Un décalage multiple de 40 donnera plutôt un effet de décalage vertical (quarante octets correspondent à une ligne).

```
1 CLEAR ,&HBF00
2 LOADM "EORTOURN
3 SCREEN 3,4,0:CLS
4 INPUT "Decalage :";DEC
5 POKE &HBF2B,DEC @ 256
6 POKE &HBF2C,DEC MOD 256
7 LOCATE 0,0,0
```

```
9 SCREEN 3,4,0:CLS
10 PSET (160,100)
20 X=160:Y=100
25 FOR I=0 TO 24
32 X=X+8*I*(-1)^I:LINE -(X,Y)
33 Y=Y+8*I*(-1)^I:LINE -(X,Y)
40 NEXT I
50 EXEC &HBF00
```

```
BF00                        ORG     $BF00

BF00 B6    E7C3             LDA     $E7C3
BF03 8A    01               ORA     #1
BF05 B7    E7C3             STA     $E7C3


BF08 FC    BF2B             LDD     DEC
BF0B 8E    4000             LDX     #$4000
BF0E 31    8B               LEAY    D,X

BF10 A6    84    B1         LDA     ,X
BF12 A8    A0               EORA    ,Y+
BF14 A7    80               STA     ,X+
BF16 108C  5F40             CMPY    #$5F40
BF1A 25    04               BLO     E1
BF1C 31    A8 E0C0              LEAY   -$1F40,Y
BF20 8C    5F40 E1          CMPX    #$5F40
BF23 25    EB               BLO     B1
BF25 30    89 E0C0              LEAX   -$1F40,X
BF29 20    E5               BRA     B1

BF2B            DEC         RMB     2
```

Une modification utile consisterait à tester le clavier avant de recommencer. Il suffirait pour cela que la routine se termine après l'instruction BLO B1 par un RTS qui rendrait la main au BASIC. Dans la deuxième version du programme, par exemple, il suffit de modifier les dernières données ou de remplacer le 48 par 63 (&H39 : RTS), puis on ajoute les lignes :

```
60 A$=INKEY$:IF LEN(A$)=0 THEN 50
70 IF A$="D" THEN 4
75 IF A$="F" THEN END
80 GOTO 50
```

Deuxième version

Cette version du programme évite d'avoir à charger le sous-programme binaire séparément.

```
1 DATA 182,231,195,138,1,183,231,195,252
,191,43,142,64,0
2 DATA 49,139,166,132,168,160,167,128,16
,140,95,64,37,4,49,169,224,192,140,95,64
,37,235,48,137,224,192,32,229
3 CLEAR,&HBEFF
4 FOR M=&HBF00 TO &HBF2A:READ D:POKE M,D
:NEXT M
5 SCREEN 3,4,0:CLS
6 INPUT "Decalage :";DEC
7 POKE &HBF2B,DEC @ 256
8 POKE &HBF2C,DEC MOD 256
9 CLS:LOCATE 0,0,0
10 PSET (160,100)
20 X=160:Y=100
25 FOR I=0 TO 24
32 X=X+8*1#(-1)^1:LINE -(X,Y)
33 Y=Y+8*1#(-1)^1:LINE -(X,Y)
40 NEXT I
50 EXEC &HBF00
```

Décalage

Ce programme crée un curieux défilement de tout ce qui est affiché à l'écran.

```
                E7C3    PRC     EQU     $E7C3
                4000    PAGDEB  EQU     $4000
                5F40    PAGFIN  EQU     $5F40
                5F3E    PAGDER  EQU     $5F3E

BF00                            ORG     $BF00

                        * Travail sur les form
BF00 B6 E7C3                    LDA     PRC
BF03 8A 01                      ORA     #1
BF05 B7 E7C3                    STA     PRC
                        * On prend 16 points
                        * que l'on decale...
```

```
BF08 FC   4000     B1      LDD     PAGDEB
BF0B 34   06               PSHS    D
BF0D 8E   4000             LDX     #PAGDEB
BF10 EC   02       B2      LDD     2,X
BF12 ED   81               STD     ,X++
BF14 8C   5F40             CMPX    #PAGFIN
BF17 26   F7               BNE     B2
BF19 35   06               PULS    D
BF1B FD   5F3E             STD     PAGDER
BF1E 20   E8               BRA     B1

          0000             END

00000 Total Errors
```

Nettoyage

Voici un programme d'initialisation de l'écran (il fait la même chose que SCREEN 3,4 : CLS ou $DC = 11Ø111ØØ).

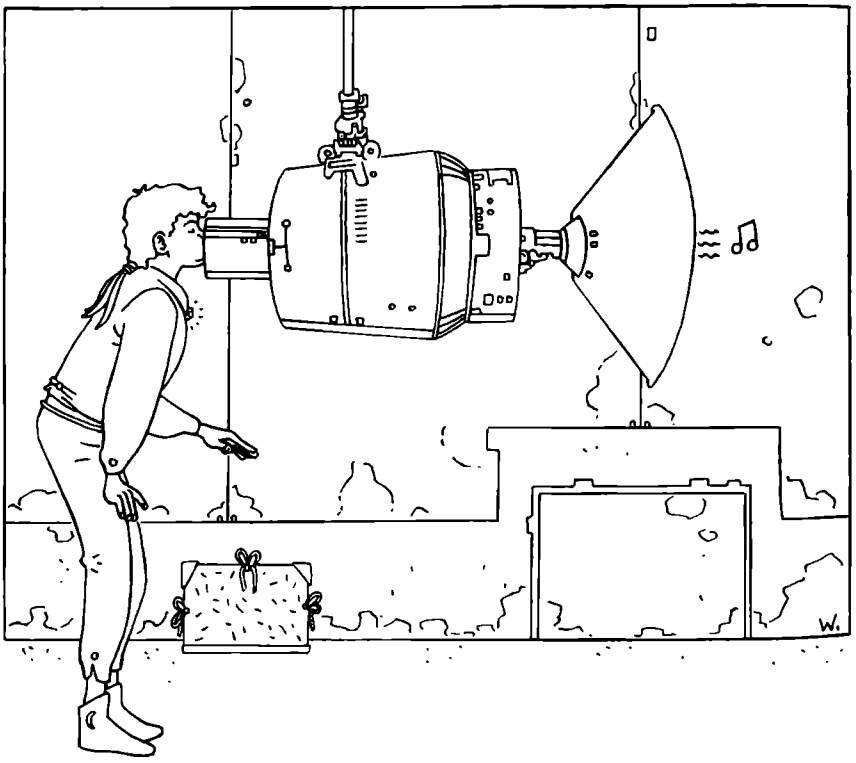
```
B000                       ORG     $B000

B000 B6   E7C3             LDA     $E7C3
B003 84   FE               ANDA    #$FE
B005 B7   E7C3             STA     $E7C3
B008 86   DC               LDA     #$DC
B00A BD   B017             JSR     NET

B00D B6   E7C3             LDA     $E7C3
B010 8A   01               ORA     #1
B012 B7   E7C3             STA     $E7C3
B015 86   00               LDA     #0

B017 8E   4000   NET       LDX     #$4000
B01A A7   80     B0        STA     ,X+
B01C 8C   5F40             CMPX    #$5F40
B01F 26   F9               BNE     B0
B021 39                    RTS

          0000             END

00000 Total Errors
```

Interlude

Recopie d'écran sur imprimante à impact

```
0 ' Copie d'ecran sur PR90080
1 GOTO 3:' EXEC (M0)
2 FOR M0=&HDF70 TO &H7F70 STEP -&H2000:
POKE M0,85:IF PEEK(M0)=85 THEN RETURN
ELSE NEXT:RETURN:' Trouve le haut de la
RAM
3 GOSUB 2:CLEAR,M0-1:GOSUB 2:FOR M=M0 TO
 M0+143:READ D:POKE M,D:NEXT M
4 DATA 52,54,198,64,247,96,43,189,232,18
,37,120,182,231,195,138,1,183,231,195,14
2,95,64,134,120,111,128,74,38,251,198,8,
247,96,43,189,232,18,37,92,142,64,0,134,
40,52,2,52
5 DATA 16,134,8,52,2,198,64,166,228,52,2
,166,132,68,106,228,38,251,50,97,48,136,
40,86,36,237,86,189,232,18,37,52,48,137,
254,232,106,228,38,221,50,97,53,16,48,1,
106,228
6 DATA 38,205,50,97,198,10,189,232,18,37
,25,48,137,0,240,140,95,64,37,183,204,4,
10,189,232,18,37,8,74,38,248,198,15,189,
232,18,52,1,198,16,247,96,43,189,232,18,
53,183
```

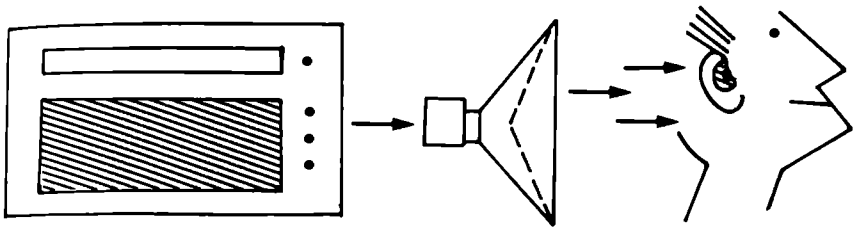
Bruits



Bzzs, couacs et pouêts

Le TO7 fait du bruit, c'est indéniable. Il en fait chaque fois que l'on appuie sur une touche, il en fait également quand on utilise son générateur de musique, soit à partir d'un langage comme le BASIC : PLAY « 02A1T2ØMIMIMIDO », soit à partir d'un programme spécialisé comme Symphonia ou Mélodia.

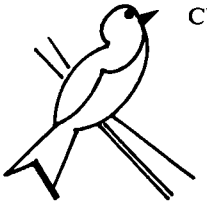
Tout d'abord, mettons-nous bien d'accord sur ce qu'est un son. Un son est une vibration de l'air.



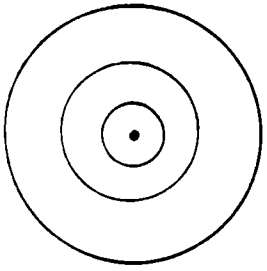
L'air vibre

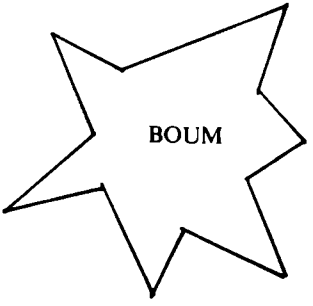
la membrane
du haut parleur
agite l'air

la membrane
du typan
agite les nerfs



CUI CUI CUI

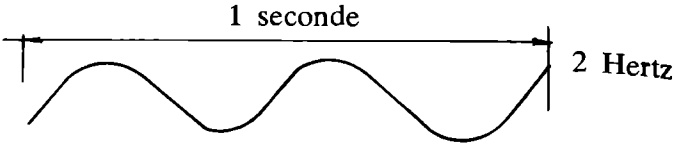




un peu comme les vagues crées
par un caillou jetté dans l'eau...

La fréquence : c'est le paramètre qui permet de mesurer la hauteur des sons.

Une basse fréquence donne un son grave :



Une fréquence élevée donne un son aigu :

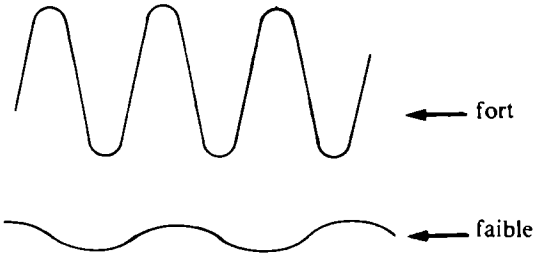
L'unité de mesure de la fréquence est le Hertz. C'est le nombre d'oscillations par seconde. Le « la » de référence vibre à 435 Hertz, soit 435 fois par seconde. L'oreille perçoit les sons dont les fréquences sont comprises entre 30 Hertz pour les plus graves et 30 000 Hertz pour les plus aigus. En deçà et au-delà de ces limites, on parle d'infrasons et d'ultrasons.

La durée d'une oscillation est appelée période. Plus la fréquence est grande, plus la période est petite. Précisément, on a :

$$P = \frac{1}{F} \quad \text{ou} \quad F = \frac{1}{P}$$

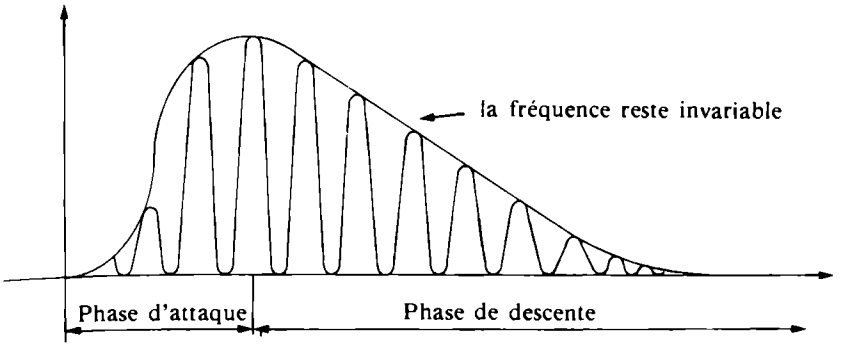
La période du « la », 435 Hertz, dure 1/435 = environ 0,002 secondes, soit environ 2×10^{-3} secondes, ou deux millièmes de seconde, ou deux millisecondes[1].

L'intensité : c'est le paramètre qui permet de mesurer le volume des sons. Il correspond à une plus ou moins grande amplitude des oscillations.



1. Le découpage des secondes : 1/1 000 de seconde = 10^{-3} ou une milliseconde; 1/1 000 000 de seconde = 10^{-6} secondes ou un millionième de seconde ou une microseconde; 1/ 1 000 000 000 de seconde = 10^{-9} secondes ou un milliardième de seconde ou une nanoseconde.

Dans certains sons, l'intensité peut varier. Un son de cloche sera perçu comme ayant une intensité décroissante jusqu'à ce que l'oreille n'entende plus rien après une attaque très brève (le coup de cloche).



Ces variations d'intensité dans la vie d'un son sont enfermées dans une courbe appelée l'enveloppe du son.

L'harmonie : le signal périodique qui produit un son peut être plus ou moins complexe. C'est la forme des oscillations ainsi que l'enveloppe qui font que le même « la » 435 Hz produit par un Stradivarius ou par la Castafiore n'est pas perçu de manière identique par l'oreille.

signal complexe
quelques signaux

signal sinusoïdal

signal carré

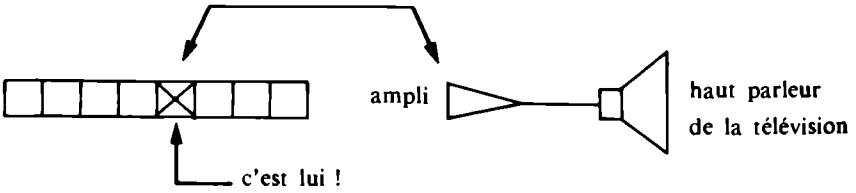
Le mathématicien J.-B. Fourier a montré que tout signal périodique peut être obtenu en additionnant des signaux sinusoïdaux de fréquences et d'amplitudes diverses. Ce résultat extraordinaire est à la base de tous les systèmes de synthèse de son.

Bip, bip

Nous allons considérablement simplifier tout cela. Nous allons travailler exclusivement sur la fréquence : les sons que nous allons fabriquer sont tous des signaux carrés.

Sans interface spécialisée, la seule possibilité que nous ayons est de faire varier le son en tout ou rien, 1 ou Ø.

L'un des huit bits de l'un des 65 536 octets de l'espace adressable par le 68Ø9 de notre TO7 est en fait relié au buzzer et à la ligne son de la prise Péritel du téléviseur. Ce bit, comme ses confrères, peut être à 1 ou à Ø. Une variation plus ou moins rapide de la valeur de ce bit produira un son plus ou moins aigu. Ce bit est le quatrième bit du port d'entrée-sortie C dont l'adresse est $E7C1.



Voici un petit programme en binaire qui permet de modifier l'état de ce bit selon une fréquence audible : c'est le programme qui produit le « bip » élémentaire du clavier : 17 demi-oscillations de 580 microsecondes, soit environ dix millisecondes d'un son à 750 Hertz.

Les instructions élémentaires du microprocesseur sont exécutées en un temps parfaitement déterminé. Le microprocesseur est piloté par une horloge à 1 MégaHertz. C'est dire que son cycle élémentaire est exécuté en un millionième de seconde ($1Ø^{-6}$s), que l'on écrit 1μs). Une instruction comme EORA #$Ø8 qui sera codée sur deux octets, 88 Ø8, sera exécutée en deux cycles, c'est-à-dire en 2 μs (voir annexe 1, tableaux, p.158).

```
                E7C1    CRC     EQU     $E7C1

7000                            ORG     $7000


7000 C6    11       BZZ     LDB     #17
7002 B6    E7C1     BIP     LDA .   CRC         5
7005 88    08              EORA    #$08        2
7007 B7    E7C1            STA     CRC         5
700A 8E    0050            LDX     #80         3

700D 30    1F       BOUCL   LEAX    -1,X        4
700F 26    FC              BNE     BOUCL       3

7011 5A                    DECB                2
7012 26    EE              BNE     BIP         3
7014 39                    RTS

                0000        END

00000 Total Errors
```

L'accumulateur B est chargé avec la valeur 17 (17 en décimal = $11). Cette valeur représente le nombre de changements d'état de la membrane du haut-parleur. C'est l'accumulateur en A qui est chargé avec le contenu de CRC (l'adresse $E7C1). Cette instruction inverse l'état du quatrième bit de l'accumulateur A.

EOR réalise un « OU exclusif » dont la table de vérité est :

b	b'	b EOR b'
1	1	Ø
1	Ø	1
Ø	1	1
Ø	Ø	Ø

On remarque que Ø EOR b = b
et que 1 EOR b = $\bar{b}$

($\bar{b}$ = non-b).

b	$\bar{b}$
1	Ø
Ø	1

$Ø8 = %ØØØØ1ØØØ

STA CRC

Le contenu de A avec le quatrième bit modifié est rangé dans CRC. La ligne du haut-parleur change d'état : si elle était à Ø, elle passe à 1, si elle était à 1, elle passe à Ø.

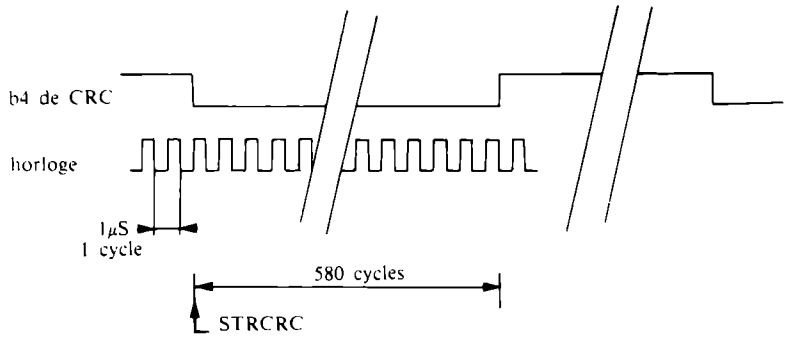
LDX #8Ø

Initialisation de la boucle d'attente : on charge le registre avec 8Ø =($5Ø). Le programme va tourner 8Ø fois dans les deux instructions suivantes simplement pour perdre du temps. De cette valeur dépend la fréquence de la note. On obtiendra une note plus grave avec une valeur plus grande.

Boucle d'attente : on diminue la valeur contenue dans le registre X de 1. Tant que cette valeur n'est pas nulle, on recommence. On diminue la valeur contenue dans le registre B de 1. Tant que cette valeur n'est pas nulle on reprend à partir de la seconde instruction. Si le contenu de B devient nul, c'est la fin du programme.

L'horloge du microprocesseur bat à 1 MégaHertz (10^6 Hz). Le cycle élémentaire dure donc une microseconde (1 μs = 10^{-6}s). En face de chaque instruction, on a donné le nombre de cycles que met cette instruction pour être exécutée. La boucle d'attente représente (4 + 3) x 80 = 560 cycles et les quelques autres instructions sont exécutées en 20 cycles. La période de notre son est donc de 2 x 580 soit 1060 cycles. La fréquence du son est donc :

$$F = \frac{1}{1060 \times 10^{-6}} \text{ ou environ } 943 \text{ Hz}$$



Le petit programme binaire suivant permet de générer un son défini par les paramètres DUR (seize bits) et FREQ (huit bits). Ces paramètres sont passés aux adresses $BØØØ à $BØØ2. Ce programme fonctionne exactement sur le principe du précédent, sauf que DUR et FREQ sont à présent variables. Une fois le programme assemblé, il est sauvé sous le nom binaire de MUSIK Ø.

```
B000                            ORG     $B000

B000                DUR         RMB     2
B002                FREQ        RMB     1


B003 BE   B000 DEB              LDX     DUR
B006 F6   B002                  LDB     FREQ

B009 34   04   DEB2             PSHS    B

B00B B6   E7C1 B0               LDA     $E7C1
B00E 88   08                    EORA    #8
B010 B7   E7C1                  STA     $E7C1

B013 5A              B1         DECB
B014 26   FD                    BNE     B1

B016 35   04                    PULS    B

B018 30   1F                    LEAX    -1,X
B01A 26   ED                    BNE     DEB2
B01C 39                         RTS



          0000                  END

00000 Total Errors



B0          B00B
B1          B013
DEB         B003
DEB2        B009
DUR         B000
FREQ        B002
```

Truc 1

Pour intégrer le programme binaire générateur de son à un programme BASIC, j'ai chargé le programme binaire, puis à l'aide d'une boucle FOR NEXT en mode direct, j'ai fait lister le code objet de ce programme. A l'aide de l'éditeur, j'en ai fait une ligne BASIC de DATA.

```
LOADM "MUSIK0
OK

FOR M=&HB003 TO &HB01C:PRINT HEX$(PEEK(M
));",";:NEXT M
BE,B0,0,F6,B0,2,34,4,B6,E7,C1,88,8,B7,E7
,C1,5A,26,FD,35,4,30,1F,26,ED,39,
OK

6 FOR M=&HB003 TO &HB01C:READ D$:POKE M,
VAL("&H"+D$):NEXT M
5 DATA BE,B0,0,F6,B0,2,34,4,B6,E7,C1,88,
8,B7,E7,C1,5A,26,FD,35,4,30,1F,26,ED,39

1 CLEAR ,&HAFFF
5 DATA BE,B0,0,F6,B0,2,34,4,B6,E7,C1,88,
8,B7,E7,C1,5A,26,FD,35,4,30,1F,26,ED,39
6 FOR M=&HB003 TO &HB01C:READ D$:POKE M,
VAL("&H"+D$):NEXT M
7 '
8 '
10 CLS:SCREEN 3,4,0
15 LOCATE 12,12,0:ATTRB 1,1
16 PRINT "Musique"
20 INPEN X,Y:IF X<0 THEN 20
25 POKE &HB000,X @ 256
26 POKE &HB001,X MOD 256
30 POKE &HB002,Y+1
35 EXEC &HB003
40 GOTO 20
```

Produit fini !

Truc 2

Le programme binaire MUSIK Ø est translatable, c'est-à-dire qu'il est possible de le faire tourner à n'importe quelle adresse. Ceci est dû au fait qu'il ne comporte pas de sauts à des adresses fixes (JMP ou JSR) mais uniquement des branchements relatifs (BNE). On

peut alors envisager une seconde technique d'intégration qui consiste à créer une chaîne de caractères dont chaque caractère est l'un des octets du code créé. Cette chaîne provoquerait sûrement les plus étranges résultats si l'on cherchait à l'afficher. Nous nous contenterons de l'exécuter !

Pour l'exécuter, nous utiliserons VARPTR qui, comme par un fait exprès, pointe sur le premier octet du code EXEC (VARPTR(M$)). Esthétique, non ? De plus, il n'y a plus besoin de réserver de zone mémoire pour le binaire : plus de CLEAR !

Cette technique marche pour les programmes translatables exclusivement et comportant moins de 256 octets; si l'on voulait gérer un programme plus gros, il faudrait sans doute utiliser un tableau. Rassurez-vous, un programme de 256 octets, ce n'est déjà pas mal.

```
5 DATA BE,B0,0,F6,B0,2,34,4,B6,E7,C1,88,
8,B7,E7,C1,5A,26,FD,35,4,30,1F,26,ED,39
6 M$="":FOR M=1 TO 26:READ D$:M$=M$+CHR$
(VAL("&H"+D$)):NEXT M
10 CLS:SCREEN 3,4,0
15 LOCATE 12,12,0:ATTRB 1,1
16 PRINT "Musique"
18 VP=VARPTR(M$):AD=256*PEEK(VP+1)+PEEK(VP+2)
20 INPEN X,Y:IF X<0 THEN 20
25 POKE &HB000,X \ 256
26 POKE &HB001,X MOD 256
27 POKE &HB002,Y+1
35 EXEC AD
40 GOTO 20
```

Si l'on voulait passer directement DUR et FREQ à des adresses non fixes, ce serait plus dur ! (Voir USR.)

Truc 3

Très osé mais vraiment amusant. Si l'on n'a plus l'intention de modifier quoi que ce soit dans le programme BASIC, on peut aller mettre le code binaire dans l'image d'une ligne de programme (en général une REMarque), entre le ØØ de fin de ligne et le début de la ligne suivante en jouant habilement sur les pointeurs.

Un autre son

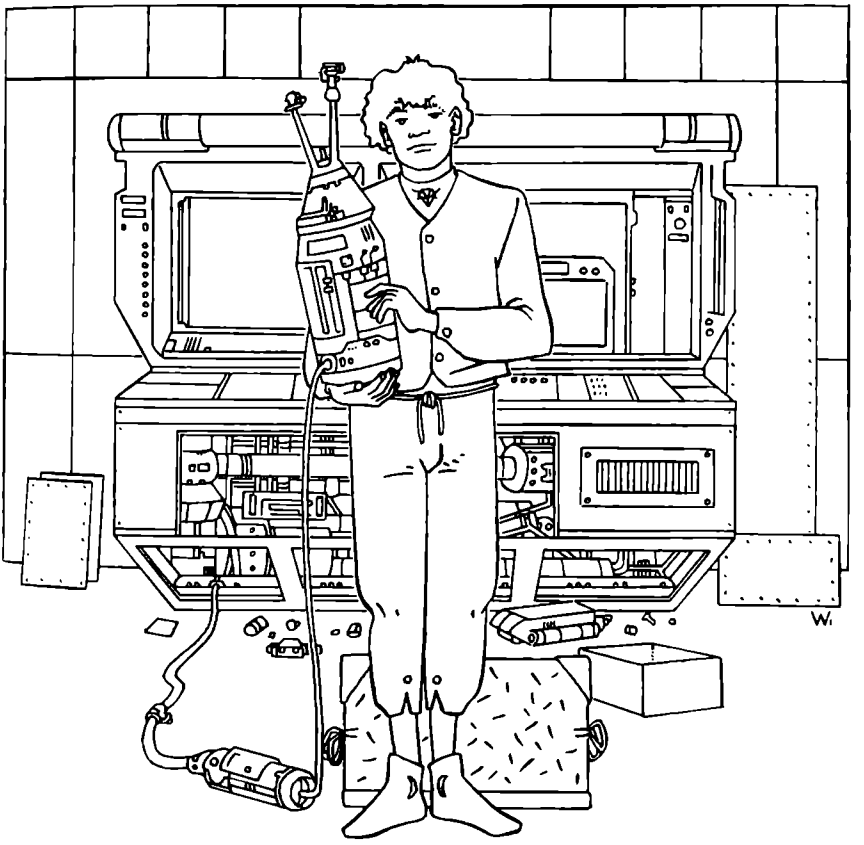
Pour créer des sons plus complexes, on peut modifier la longueur de la boucle d'attente B1 entre deux transitions successives. Dans

cet exemple, la modification est assez complexe, la durée de la boucle est calculée à partir de la valeur de FREQ en réalisant à chaque demi-période un « OU exclusif » entre FREQ et les huit bits de poids faible du compteur de durée ! (N'importe quoi, certes, mais cela vaut le coup d'être entendu !)

```
B000                        ORG     $B000

B000              DUR       RMB     2
B002              FREQ      RMB     1

B003 BE    B000   DEB       LDX     DUR

B006 F6    B002              LDB     FREQ

B009 34    04     DEB2      PSHS    B

B00B B6    E7C1   B0        LDA     $E7C1
B00E 88    08                EORA    #8
B010 B7    E7C1              STA     $E7C1
B013 5A               B1     DECB
B014 26    FD                BNE     B1
B016 35    04                PULS    B
B018 34    10                PSHS    X
B01A E8    61                EORB    1,S
B01C 35    10                PULS    X
B01E 30    1F                LEAX    -1,X
B020 26    E7                BNE     DEB2
B022 39                      RTS


           0000              END

00000 Total Errors
```

```
1 CLEAR .&HB000
5 DATA BE.B0.0.F6.B0.2.34.4.B6.E7.C1.88.
8.B7.E7.C1.5A.26.FD.35.4.34.10.E8.61.35.
10.30.1F.26.E7.39
6 FOR M=&HB003 TO &HB022:READ D$:POKE M.
VAL("&H"+D$):NEXT M
7 '
8 '
10 POKE &HB000.0:POKE &HB001.255
15 FOR S=0 TO 255
20 POKE &HB002.S
25 EXEC &HB003
30 NEXT S
OK
```

Pia, pia, pia !

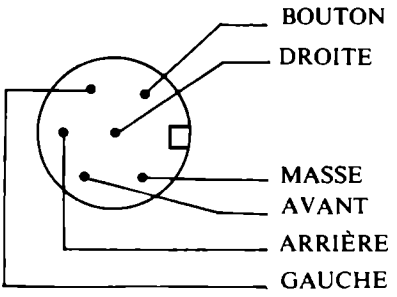


L'interface des jeux

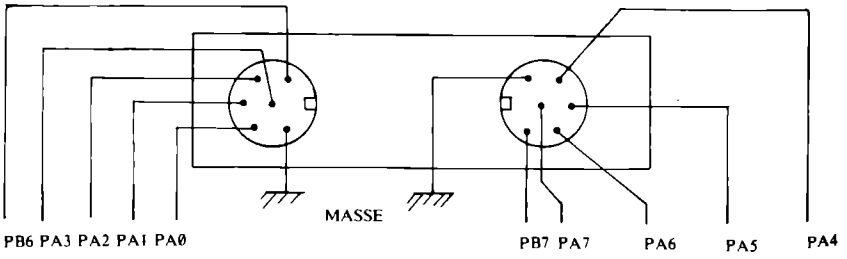
$E7CC PRA/DDRA
$E7CE CRA
$E7CD PRB/DDRB
$E7CF CRB

L'interface des jeux est principalement constituée d'une PIA 6821 dont le port A est relié aux deux fiches à six broches et le port B à un convertisseur digital-analogique dont la sortie est reliée à la patte 3B (son) du connecteur d'extensions (deux lignes du port B sont également reliées aux fiches six broches). Mis à part son utilisation conforme à sa vocation première de gestion des manettes de jeu, on peut utiliser ce circuit de très nombreuses manières : principalement, pour télécommander beaucoup de choses.

Les connexions



Vue arrière



PRA : port d'entrée-sortie A
DDRA : registre de direction des échanges du port A
CRA : registre de commande du port A

Pour utiliser l'interface en entrée

On peut utiliser les instructions BASIC STICK et STRIG; on peut également aller lire directement l'état du port A et du port B afin de voir si les différentes entrées sont ou non reliées à la masse.

```
5 CLS
10 POKE &HE7CE,0:' Sélection de DDRA
11 POKE &HE7CC,0:' Port A en entrée
12 POKE &HE7CE,4:' Sélection de PRA
13 POKE &HE7CF,0:' Sélection de DDRB
14 POKE &HE7CD,&H3F:' b7 et b6 en entrée

15 POKE &HE7CF,4:' Sélection de PRB
20 LOCATE 0,0
21 FOR L=0 TO 7
22 L$="ouverte"
23 IF PEEK(&HE7CC) AND 2^L=0 THEN L$="fe
rmée "
30 PRINT "Ligne";L;L$
35 NEXT L
40 FOR L=6 TO 7
41 L$="ouverte"
45 IF PEEK(&HE7CD) AND 2^L=0 THEN L$="fe
rmée "
50 PRINT "Ligne";L+2;L$
55 NEXT L
60 GOTO 20
```

Ce programme n'a évidemment que peu d'intérêt par lui-même mais on comprendra qu'il peut être sensiblement amélioré, le TO7 pouvant ainsi « palper » l'état d'un certain nombre de contacts réalisés entre les différents plots PA et PB et la masse commune.

Pour utiliser l'interface en sortie

Lorsque l'on utilise le PIA en sortie, il est important de savoir que l'on ne peut pas « tirer » beaucoup de courant des lignes PA et PB. On risque en effet de détruire le PIA. Le 6821 n'est pas un circuit cher (une vingtaine de francs) mais son remplacement est très pénible car il est soudé ! On aura donc soin de ne pas tirer plus d'une charge TTL sur ces lignes.

```
POKE &HE7CE,0:'    sélection de DDRA
POKE &HE7CC,&HFF:' PA en sortie
POKE &HE7CE,4:'    sélection de PRA
```

On peut à présent programmer les différentes lignes du port A par des instructions du type :

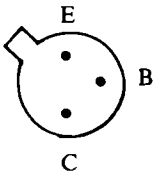
```
POKE &HE7CC,PEEK(&HE7CC) XOR 2^L
```

qui font basculer la ligne L. On peut bien entendu utiliser AND ou OR pour forcer à Ø ou à 1 certaines lignes.

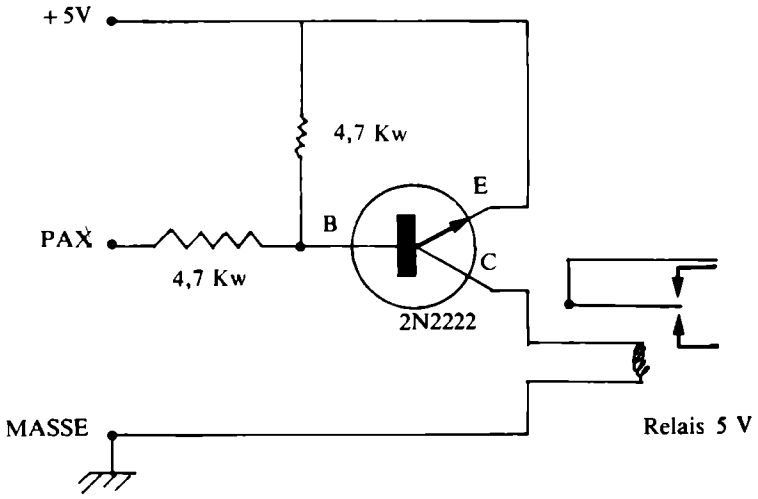
Bien évidemment il est possible de panacher les différentes lignes en entrée et en sortie en initialisant DDRA :

```
POKE &HE7CE,Ø
POKE &HE700,&HFØ
```

initialise PA7, PA6, PA5 et PA4 en sortie alors que PA3, PA2, PA1 et PAØ seront en entrée.



vue de dessous
du 2N2222



Voici un petit montage qui s'adapte délicieusement bien à l'une quelconque des sorties PA. On aura tout intérêt à aller chercher le + 5V à l'intérieur de l'interface sur la broche 19A du connecteur.

Il peut être utile de savoir que rien n'empêche d'ouvrir l'interface à l'aide d'un tournevis Parker pour constater que les deux lignes CB1 et CB2 ne sont pas connectées. Il est possible de programmer la

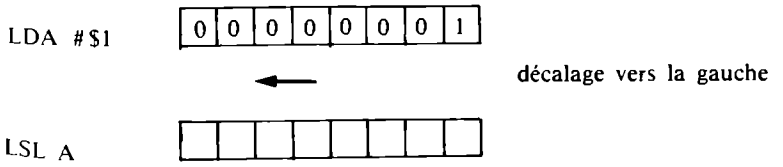
ligne CB2 en sortie : on peut alors tirer sur cette ligne 1 mA sous 1,5 V, ce qui est en général très suffisant pour commander directement un transistor de commutation.

Pour utiliser cette ligne, il faut forcer à 1 les b4 et b5 de CRB, l'état de la ligne CB2 reflète alors l'état du b3 de CRB (de plus, le b2 de CRB est en général 1 afin de sélectionner PRB plutôt que DDRB à l'adresse &E7CD). On manipule donc les deux instructions suivantes :
POKE &HE7CF, &H34 (mise à Ø de CB2)
POKE &HE7CF, &H3C (mise à 1 de CB2)
Le programmeur avancé saura sûrement tirer parti de la gestion des lignes d'interruption mais tout cela vous paraîtra simple et facile à mettre en œuvre lorsque vous ressentirez le besoin de vous en servir.

Musique (bis)

Les six lignes de poids faible du port B (PbØ - Pb5) sont reliées à un convertisseur digital-analogique (DAC). En fonction du mot envoyé sur ces six lignes, le DAC sort une tension déterminée sur la ligne son de la prise Péritel. Il est ainsi possible de créer des signaux très sensiblement plus complexes que ceux réalisés en tout ou rien. On peut par exemple créer des tables de formes d'onde (à l'aide d'une somme de fonctions trigonométriques, par exemple) afin de charger PRB avec des valeurs déterminées.

Dans le programme suivant, la ligne 2Ø contient l'image du programme assembleur donné plus loin. La forme d'onde est déterminée très simplement par les instructions



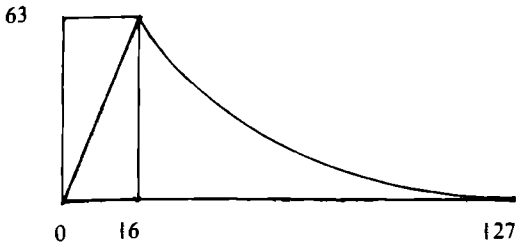

```
10 CLEAR &HB7FF
20 DATA 134,0,183,231,207,134,255,183,23
1,205,134,52,183,231,207,190,184,0,184,1
,72,39,251,183,231,205,246,184,2,90,38,2
53,48,31,38,240,57
30 FOR M=&HB800 TO &HB827:READ D:POKE M,
D:NEXT M
```

```
50 POKE &HB800,0
51 POKE &HB801,255*RND
52 POKE &HB802,255*RND
55 EXEC &HB803
60 GOTO 50
OK
```

```
            E7CD    PRB     EQU     $E7CD
            E7CD    DDRB    EQU     $E7CD
            E7CF    CRB     EQU     $E7CF

B800                        ORG     $B800

B800                DUR     RMB     2
B802                FREQ    RMB     1
                    * Initialisation PIA
B803 86     00      DEB     LDA     #0
B805 B7     E7CF            STA     CRB
                    * Lignes en sortie
B808 86     FF              LDA     #$FF
B80A B7     E7CD            STA     DDRB
                    * Validation de PRB
B80D 86     34              LDA     #$34
B80F B7     E7CF            STA     CRB
                    * Un son...
B812 BE     B800            LDX     DUR
B815 86     01      A1      LDA     #$1
B817 48             BOU     LSLA
B818 27     FB              BEQ     A1
B81A B7     E7CD            STA     PRB
B81D F6     B802            LDB     FREQ
B820 5A             B1      DECB
B821 26     FD              BNE     B1
B823 30     1F              LEAX    -1,X
B825 26     F0              BNE     BOU

B827 39                     RTS


            0000            END

00000 Total Errors
```

Si l'on souhaite une forme d'onde plus complexe, on peut créer une table qui sera consultée régulièrement lors de la « vie » du son.

Voici un exemple permettant de créer et d'utiliser une table de forme d'onde de 128 octets.



En se fatiguant un peu, il devrait être possible de réaliser un programme qui exploite une forme d'onde dessinée au crayon optique sur l'écran.

Grâce à cette interface, la cartouche PHONEMIA fait don de la parole au TO7.

Greffes, USR

Souvent, on aura envie de réaliser des programmes hybrides constitués d'un programme principal en BASIC et de petits bouts de binaires qui pourront représenter tout aussi bien de petits sous-programmes assembleurs que des données (valeurs numériques, mots d'un dictionnaire, etc.) ou encore des éléments graphiques.

Les ordres BASIC, SAVEM et LOADM, permettent de réaliser des transferts de portions de mémoires vers le disque ou la cassette et réciproquement, et ils peuvent être intégrés à des programmes. Pour les utiliser correctement, il faut simplement avoir une idée très précise de la carte mémoire de la machine au moment de leur exécution. L'instruction CLEAR permet de réserver de la place, en haut de mémoire RAM, pour les programmes et données binaires. Cette instruction doit en général se trouver parmi les toutes premières d'un programme BASIC car elle détermine la répartition de la mémoire entre BASIC, les chaînes, le binaire et les caractères définissables. Une modification de cette répartition après un début de travail peut avoir de fâcheuses conséquences.

Au chapitre sur la musique, on a vu comment greffer du binaire et du BASIC de différentes manières. La transmission des paramètres entre BASIC et le programme binaire se faisait alors par l'intermédiaire d'adresse fixes, à l'aide de PEEK et POKE. Une autre technique consiste à utiliser la fonction USR.

Cette fonction BASIC permet d'appeler des sous-programmes binaires à partir de BASIC en passant éventuellement un paramètre variable de BASIC vers le programme binaire, et inversement du programme binaire vers BASIC. On commence par écrire un programme binaire résidant à une adresse donnée. Ce programme étant présent en mémoire, on peut définir une fonction USR par une instruction du style :

```
DEF USR Ø = &HBFØØ
```

L'instruction USRØ(X) sera alors manipulable en BASIC comme n'importe quelle fonction numérique ou alphanumérique telle que SIN(X) ou LEN(X$). Quand cette instruction est exécutée, BASIC passe au sous-programme binaire le type de la variable dans le registre A (2 pour une variable entière, 4 pour une variable simple précision, 8 pour une variable double précision, 3 pour une variable alphanumérique), et dans le registre X le pointeur de données (on constate d'ailleurs que l'adresse de ce pointeur est toujours 24 917, ce qui sera très utile dans le chapitre suivant).

Quand le sous-progrmme binaire a terminé son travail, il retourne une valeur dont le type est dans A et dont le descripteur est pointé par X. On peut utiliser plusieurs fonctions USR :

USRØ (le Ø est facultatif)
USR1
USR2
etc.

```
BF00                        ORG     $BF00

BF00 34   12                PSHS    A,X
BF02 B7   BF17              STA     MA
BF05 BF   BF18              STX     MX

BF08 108E BF1A              LDY     #FAC
BF0C 86   03                LDA     #3

BF0E E6   86    B0          LDB     A,X
BF10 E7   A6                STB     A,Y
BF12 4A                     DECA
BF13 2A   F9                BPL     B0

BF15 35   92                PULS    A,X,PC


BF17                MA      RMB     1
BF18                MX      RMB     2
BF1A                FAC     RMB     4

           0000             END

00000 Total Errors
```

```
B0          BF0E
FAC         BF1A
MA          BF17
MX          BF18
```

Voici un petit programme qui montre comment il est possible de passer des paramètres à un sous-programme assembleur à partir de BASIC. Il est fait pour s'intégrer à un programme BASIC.

```
Found: USR0       BAS
OK
LIST

10 DATA 34,12,B7,BF,17,BF,BF,18,10,8E,BF
,1A,86,03,E6,86,E7,A6,4A,2A,F9,35,92
20 FOR M=&HBF00 TO &HBF16
21 READ D$:D$="&H"+D$:D=VAL(D$)
22 POKE M,D
23 NEXT M
30 DEFUSR=&HBF00
40 ZZ=USR(5)
41 A=PEEK(&HBF17)
42 X=256*PEEK(&HBF18)+PEEK(&HBF19)
45 PRINT ZZ,A,X
50 FOR I=0 TO 3
55 PRINT PEEK(&HBF1A + I);
60 NEXT I

RUN
 5              2              24917
 0   0   0   5
OK

RUN
 5              2              24917
 0   0   0   5

40 ZZ=USR(D)
RUN
 146            4              24917
 136   146   0   0

40 ZZ=USR(M)
RUN
 48919          4              24917
 144   191   23   0

5 K%=12
40 ZZ=USR(K%)
RUN
 12             2              24917
 144   0   0   12
```

Dans le programme BASIC, on trouvera des instructions du type :
CLEAR &HAFFF
DEF USR = &HBØØØ
LOADM « GRAPH (GRAPH est le fichier binaire)

22 = USR (Ø) Sauvegarde de la ligne 25
22 = USR (1) Restitution de la ligne 25
22 = USR (2) Copie de la page couleur 1 sur la page couleur 2
22 = USR (3) Echange des pages couleurs 1 et 2

Voici à présent quelques sous-programmes binaires pour TO7 que l'on peut intégrer à un programme BASIC. Ils permettent en particulier d'utiliser la seconde page couleur et de sauver puis de restituer la ligne 25 pour établir un dialogue avec l'utilisateur. Ces routines ont été écrites pour s'intégrer dans un éditeur graphique.

Il faut savoir que lorsque BASIC exécute une instruction USR, l'argument de la fonction USR se trouve dans FAC dont l'adresse de début est pointée par le registre X, le type de variable étant contenu dans le registre A. Dans notre exemple, la variable passée est entière, d'où le LDA3,X. Le ASLA sert à multiplier par 2 cette variable car les adresses de la table de sauts (JMPTBL) tiennent sur deux octets.

```
**********************************************
*                                            *
*    GRAPH2      ROUTINES GRAPHIQUES         *
*                POUR LE PROGRAMME           *
*                PEINT2                      *
*                                            *
*                J.B. TOUCHARD               *
*                                            *
*                      10/01/1984            *
*                                            *
**********************************************

          E7C3     PRC     EQU     $E7C3

Buffers
B000                       ORG     $B000

Ligne 25
B000                LIBAS  RMB     640
```

```
B800                            ORG     $B800
           AF                   SETDP   $AF

B800 34    1E       DEBUT       PSHS    D,X,DP
B802 86    AF                   LDA     #$AF
B804 1F    8B                   TFR     A,DP
B806 A6    03                   LDA     3,X
B808 48                         ASLA
B809 8E    B810                 LDX     #JMPTBL
B80C AD    96                   JSR     [A,X]
B80E 35    9E                   PULS    D,X,DP,PC

B810 B818          JMPTBL       FDB     SAVBA
B812 B835                       FDB     RAMBA
B814 B852                       FDB     COPP12
B816 B868                       FDB     EXPP12
```

Sauvegarde ligne 24

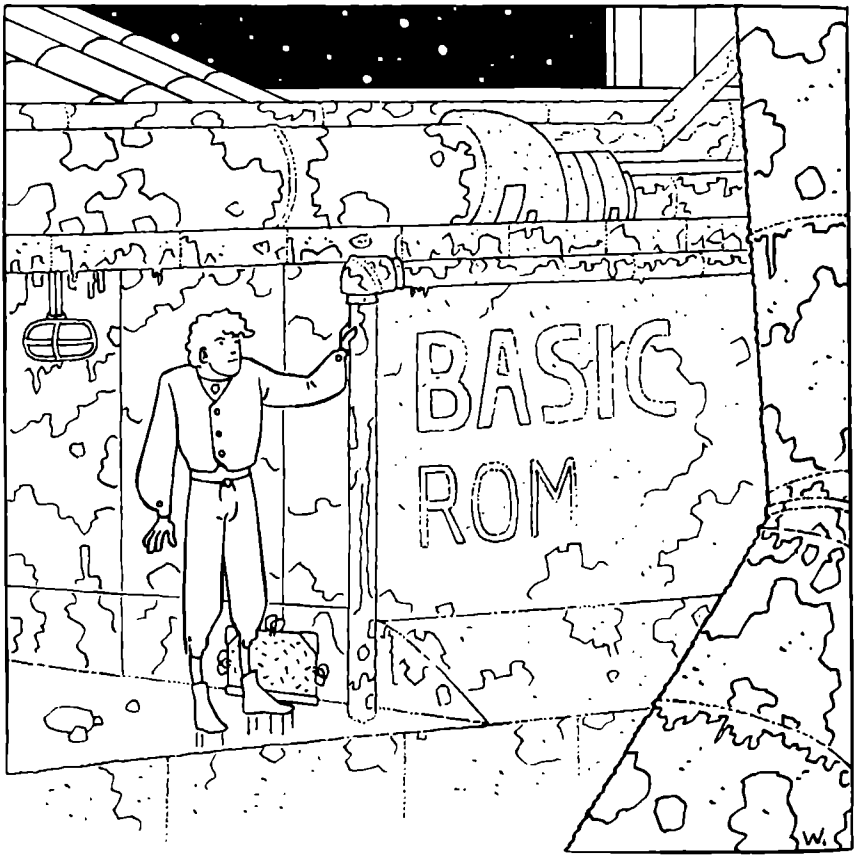
```
B818 BD    B899     SAVBA       JSR     FORM
B81B 108E  B000                 LDY     #LIBAS
B81F BD    07                   BSR     SBS
B821 7A    E7C3                 DEC     PRC
B824 108E  B140                 LDY     #LIBAS+320
B828 8E    5E00     SBS         LDX     #$5E00
B82B EC    81       SBS1        LDD     ,X++
B82D ED    A1                   STD     ,Y++
B82F 8C    5F40                 CMPX    #$5F40
B832 26    F7                   BNE     SBS1
B834 39                         RTS
```

Restitution ligne 24

```
B835 BD    B899     RAMBA       JSR     FORM
B838 108E  B000                 LDY     #LIBAS
B83C BD    07                   BSR     SBR
B83E 7A    E7C3                 DEC     PRC
B841 108E  B140                 LDY     #LIBAS+320
B845 8E    5E00     SBR         LDX     #$5E00
B848 EC    A1       SBR1        LDD     ,Y++
B84A ED    81                   STD     ,X++
B84C 8C    5F40                 CMPX    #$5F40

B84F 26    F7                   BNE     SBR1
B851 39                         RTS
```

```
Copie P1 sur P2 couleur
B852 BD   B890     COPP12 JSR    COUL
B855 8E   4000            LDX    #$4000
B858 EC   84       BCOP   LDD    ,X
B85A BD   B884            JSR    P1P2
B85D ED   81              STD    ,X++
B85F BD   B884            JSR    P1P2
B862 8C   5F40            CMPX   #$5F40
B865 26   F1              BNE    BCOP
B867 39                   RTS
Echange P1 et P2 couleur
B868 BD   B890     EXPP12 JSR    COUL
B86B 8E   4000            LDX    #$4000
B86E EC   84       BEXP   LDD    ,X
B870 BD   B884            JSR    P1P2
B873 10AE 84              LDY    ,X
B876 ED   84              STD    ,X
B878 BD   B884            JSR    P1P2
B87B 10AF 81              STY    ,X++
B87E 8C   5F40            CMPX   #$5F40
B881 26   EB              BNE    BEXP
B883 39                   RTS
echange p1p2
B884 34   02       P1P2   PSHS   A
B886 B6   E7C3            LDA    PRC
B889 88   04              EORA   #$4
B88B B7   E7C3            STA    PRC
B88E 35   82              PULS   A,PC
Ecriture couleur
B890 B6   E7C3     COUL   LDA    PRC
B893 84   FC              ANDA   #$FC
B895 B7   E7C3            STA    PRC
B898 39                   RTS
Ecriture forme
B899 B6   E7C3     FORM   LDA    PRC
B89C 8A   01              ORA    #1
B89E B7   E7C3            STA    PRC
B8A1 39                   RTS

          0000            END

00000 Total Errors
```

Enquête dans la ROM BASIC



A la recherche des constantes mathématiques

BASIC contient un certain nombre de « routines » binaires tout à fait intéressantes qu'il peut être bien agréable d'utiliser à partir de programmes assembleur. C'est par exemple le cas de « Math Pack ». Lorsque l'on écrit des programmes en assembleur, on peut avoir à effectuer certains calculs sur des nombres exprimés en virgule flottante.

On n'aura sans doute pas envie d'écrire les programmes qui permettent de multiplier, diviser, additionner ou soustraire ces nombres alors qu'ils résident quelque part dans les 16 K de BASIC.

Le problème est de savoir comment localiser ces programmes et comment les utiliser. Microsoft ne donne pas d'indications là-dessus, il faut se livrer à une enquête à partir des indices suivants :

- On connaît bien la structure des nombres exprimés en virgule flottante (voir chapitre « Variables »).

- On est à peu près sûr que Math Pack utilise deux registres de quatre octets pour manipuler les nombres flottants, appelons-les FAC et AUX (un seul registre pour les fonctions d'une seule variable comme SIN, LOG, etc., et deux registres pour les fonctions de deux variables, +, —, ↑, etc.). Le travail sur la fonction USR semble montrer que FAC se trouve à l'adresse $6155-$615C.

- On sait que les méthodes pour calculer en virgule flottante font appel à un certain nombre de constantes comme $\pi/2$ (pour la trigonométrie : COS est sûrement calculé à partir de SIN par l'égalité COS(T) = SIN ($\pi/2$ + T)), et comme 1/2 (pour SQR : SQR(X) est sûrement calculé à partir de la fonction ↑ par l'égalité : SQR(X) = X ↑ (1/2)). Pour des raisons de traduction de base 2 en base 10, on peut également espérer localiser des constantes comme LOG2, $\log_2(e)$, la multiplication par 10 étant également importante, on peut espérer localiser la constante 10 du côté de la multiplication.

```
10 A=3.14159265/2
20 PT=VARPTR(A)
25 A0=PEEK(PT)
26 A1=PEEK(PT+1)
27 A2=PEEK(PT+2)
28 A3=PEEK(PT+3)
30 FOR M=0 TO &H3FFF
35 IF PEEK(M)<>A0 THEN 50
36 IF PEEK(M+1)<>A1 THEN 50
37 IF PEEK(M+2)<>A2 THEN 50
38 IF PEEK(M+3)<>A3 THEN 50
40 PRINT HEX$(M)
50 NEXT M
```

Ce petit programme, modifié à l'occasion, permet de balayer la mémoire ROM BASIC à la recherche de la constante déclarée en ligne 1Ø. Il faut naturellement que la précision donnée à cette constante soit égale ou meilleure que celle que connaît BASIC. En lignes 25 à 28, on sort les valeurs contenues par les quatre octets qui représentent cette constante, et on recherche ensuite quatre octets consécutifs dans la ROM qui contiennent ces valeurs.

Voici les résultats obtenus :

Constante	Image en mémoire (HEXA)	Trouvée à l'adresse
PI/2	81, 49, ØF, DB	26FD
SQR (1/2)	8Ø, 35, Ø4, F3	19DØ
SQR (2)	81, 35, Ø4, F3	19D4
LOG (2)	8Ø, 31, 72, 18	19DC
$\log_2$ (e)	81, 38, AA, 3B	23D6
1/2	8Ø, ØØ, ØØ, ØØ	2334 238Ø
1Ø	84, 2Ø, ØØ, ØØ	1B1B

A l'aventure dans le Math Pack

En modifiant légèrement le programme ci-dessus, on recherche alors où ces constantes sont utilisées (à peu près certainement par une instruction assembleur de type LDX #26FD) dont le code est 8E, 26, FD, qui sert à « pointer » sur la constante pi/2).

```
25 A0=&H26
26 A1=&HFD
```

```
30 FOR M=0 TO &H3FFF
35 IF PEEK(M)<>A0 THEN 50
36 IF PEEK(M+1)<>A1 THEN 50
40 PRINT HEX$(M)
50 NEXT M
```

On obtient ceci :

Constante	Trouvée en	Utilisée en
$\pi/2$	26FD	268A
SQR (1/2)	19DØ	19E9
SQR (2)	19D4	19F9
LOG (2)	19DC	1A22
$\log_2$ (e)	23D6	1DØB
		23FF
1/2	2334	2713
	238Ø	184B
		238C
1Ø	1B1B	1B2C

Les résultats précédents permettent de se faire une bonne idée de l'endroit où réside le Math Pack. Il faut à présent se lancer dans le désassemblage du BASIC. Commençons par exemple par les fonctions SIN, COS, etc., qui ont l'avantage de n'avoir qu'un seul argument. Voici le résultat du programme PEEK, à partir de l'adresse &H268Ø.

```
2686   7E 1C A6 8E 26 FD BD 18
268E   58 96 55 81 77 25 9F BD
2696   1C 81 8E 27 01 D6 6B BD
269E   1B 2F BD 1C 81 BD 1D 71

2686   7E  1CA6      JMP    $1CA6
2689   8E  26FD      LDX    #$26FD
268C   BD  1858      JSR    $1858
268F   96  55        LDA    <$55
2691   81  77        CMPA   #$77
2693   25  9F        BCS    $2634
2695   BD  1C81      JSR    $1C81
```

Fin de la routine précédente (Notre indice !)

Pour arriver à ce résultat, on recherche dans la table des codes instructions 6809 la traduction des codes et le nombre d'octets pris par ces codes (colonne #). Le code précédent, 26FD, est 8E (LDX immédiat sur trois codes), on décode donc 8E, 26, FD en LDX #26FD. Le code suivant est alors BD, la table nous signale que cette instruction est un JSR codé sur trois octets, d'où le JSR $1858, etc.

Si notre vision du début des routines trigonométriques est exacte (COS (X) = SIN (x + pi/2)), on peut penser que $1858 réalise une addition de PI/2 avec ce qui est dans FAC.

```
1858  BD1ABF      JSR    $1ABF
185B  5D          TSTB
185C  10270404    LBEQ   $1C64
1860  8E6163      LDX    #$6163
1863  1F89        TFR    A,B
1865  4D          TSTA
1866  2601        BNE    $1869
1868  39          RTS
1869  D055        SUBB   <$55
186B  27D7        BEQ    $1844
```

Un coup d'œil vers 1ABF

```
1ABF  A601        LDA     1,X
1AC1  976B        STA    <$6B
1AC3  1F89        TFR    A,B
1AC5  8A80        ORA    #$80
1AC7  9764        STA    <$64
1AC9  D85D        EORB   <$5D
1ACB  D76C        STB    <$6C
1ACD  EE02        LDU     2,X
1ACF  DF65        STU    <$65
1AD1  9603        LDA    <$3
1AD3  2708        BEQ    $1ADD
1AD5  EE04        LDU     4,X
1AD7  DF67        STU    <$67
1AD9  EE06        LDU     6,X
1ADB  DF69        STU    <$69
1ADD  A684        LDA    ,X
1ADF  9763        STA    <$63
1AE1  D655        LDB    <$55
1AE3  39          RTS
```

Quelques remarques sur le décodage : l'instruction A6 comporte un « post-octet » Ø1. On le décode grâce à la table suivante qui permet de décoder les adressages indexés. L'instruction IFTFR se décode de manière analogue, grâce à la table suivante qui permet de savoir quel registre est transféré et dans quel registre il va.

SSSSDDDD	S: Source D: Destination
0000 : D	0101 : PC
0001 : X	1000 : A
0010 : Y	1001 : B
0011 : U	1010 : CC
0100 : S	1011 : DP

Exemple: "1F A2" se lit : TFR A,Y

Cette routine 1ABF mérite un peu de considération. Elle semble recopier quatre octets ou huit octets pointés par X dans un registre de quatre ou huit octets démarrant à 6163, le bit 7 de 6164 est « forcé à 1 » (ORA #$8Ø) alors que le signe du nombre a été sauvegardé dans 616B (LDA 1,X STA $616B).

Il semble que l'on ait identifié le drapeau qui permet de dire la nature du nombre traité : si 61Ø3 est nul, c'est une simple précision, s'il est non nul, c'est une double précision. On a par ailleurs identifié le registre ARG dans lequel sont passées les valeurs des fonctions à deux arguments. Cette routine 1ABF réalise en outre une mise au format arithmétique du nombre en garant son signe dans un registre spécial (b7 de 616B) et en rétablissant le 1 sous-entendu à droite du point décimal de la mantisse. En outre, à la sortie de cette routine, B contient FACEXP et A contient ARGEXP. Il est à peu près certain que cette routine précédera de peu les routines à plusieurs arguments. D'ailleurs, le programme de début du chapitre permet de trouver 1ABF aux adresses suivantes :

185Ø
1859
1A25
1B37

Heureusement que le programmeur de cette section n'a pas trop utilisé l'adressage relatif : on n'aurait rien vu.

On avance...

Demain, je désassemble le haut !

Revenons un peu à ce qui se passe autour de 1858 et désassemblons un peu plus haut.

```
1843  39         RTS
```
Toujours agréable pour localiser la fin de quelque chose et donc le début d'une autre...

```
1844  BD199A     JSR    $199A
1847  5F         CLRB
1848  2039       BRA    $1883
```
On verra ce que cela fait plus tard...

```
184A  8E2380     LDX    #$2380
```
1/2 + FAC ?

```
184D  2009       BRA    $1858
```
Ceci sert sûrement pour les arrondis!

```
184F  BD1ABF     JSR    $1ABF
```
A tous les coups, c'est le point d'entrée de la soustraction : on change le signe de FAC et on additionne.

```
1852  035D       COM    <$5D
1854  036C       COM    <$6C
1856  2003       BRA    $185B
1858  BD1ABF     JSR    $1ABF
```
Et ça, c'est le point d'entrée de l'addition. On sort ici si on ajoute Ø.

```
185B  5D         TSTB
185C  10270404   LBEQ   $1C64
1860  8E6163     LDX    #$6163
1863  1F89       TFR    A,B
1865  4D         TSTA
1866  2601       BNE    $1869
1868  39         RTS
1869  D055       SUBB   <$55
186B  27D7       BEQ    $1844
186D  250A       BCS    $1879
186F  9755       STA    <$55
```

```
1871   966B      LDA    <$6B
1873   975D      STA    <$5D
1875   8E6155    LDX    #$6155
1878   50        NEGB
1879   C1F8      CMPB   #$F8
187B   2FC7      BLE    $1844
187D   4F        CLRA
187E   6401      LSR     1,X
1880   BD19A6    JSR    $19A6
1883   D66C      LDB    <$6C
1885   2A15      BPL    $189C
1887   6301      COM     1,X    Selon le signe des acteurs, il
                                faut parfois soustraire pour
                                additionner!
1889   6302      COM     2,X
188B   6303      COM     3,X
188D   0D03      TST    <$3
188F   2708      BEQ    $1899
1891   6304      COM     4,X
1893   6305      COM     5,X
1895   6306      COM     6,X
1897   6307      COM     7,X
1899   43        COMA
189A   8900      ADCA   #$0
189C   976D      STA    <$6D
189E   9603      LDA    <$3    Double précision ?
18A0   2710      BEQ    $18B2
18A2   DC5B      LDD    <$5B
18A4   D96A      ADCB   <$6A
18A6   9969      ADCA   <$69
18A8   DD5B      STD    <$5B
18AA   DC59      LDD    <$59
```

A ce point, il paraît urgent de faire une expérience. La fonction USR semble déposer son contenu dans FAC. On peut donc essayer :

```
10 DEFUSR=&H2689
20 INPUT A
30 PRINT USR(A)
40 GOTO 10
OK
```

Cette fonction cosinus ne marche pas si mal ! On peut naturellement essayer la fonction sinus en &H268F de la même manière. Tout cela marche très bien. Pour les fonctions de deux variables, c'est un peu plus complexe, mais un peu (!) de travail de

désassemblage permet de localiser des « routines » bien intéressantes qui permettent de copier le contenu de mémoires dans FAC ou dans AUX avant d'attaquer des fonctions à deux arguments.

```
1 DEFUSR0=&H1C81:' Copie de FAC -> AUX
2 DEFUSR1=&H2391:' Fonction exp
3 INPUT A,B
4 ZZ=USR0(A)
5 PRINT USR1(B)
6 GOTO 3
```

Ce programme peut naturellement servir pour tester toutes les fonctions de deux variables.

Sous-programmes de calculs en flottant de Math Pack.

L'utilisation des routines à un argument suppose que FAC est en place.

L'utilisation des routines à deux arguments suppose que FAC est en place et que le registre X pointe sur ARG.

Elle suppose en outre que le registre $61Ø3 contient le type de variable (Ø si simple précision, – 1 si double précision).

$184A	1/2+FAC	→ FAC	FACEXP:	$6155
$184F	ARG–FAC	→ FAC	FACHO:	$6156
$1858	ARG+FAC	→ FAC	FACMO:	$6157
$19EØ	LOG(FAC)	→ FAC	FACLO:	$6158
$1A24	ARG*FAC	→ FAC	DFACHO:	$6159
$1B28	FAC/1Ø	→ FAC	DFACMHO:	$615A
$1B36	ARG/FAC	→ FAC	DFACMLO:	$615B
$1CBE	SGN(FAC)	→ FAC	DFACLO:	$615C
$1CD5	ABS(FAC)	→ FAC	FACSNG:	$615D
$1D61	FIX(FAC)	→ FAC		
$1D71	INT(FAC)	→ FAC	ARGEXP:	$6163
$2388	SQR(FAC)	→ FAC	....	
$2391	ARG^FAC	→ FAC	ARGSNG:	$616B
$23FB	EXP(FAC)	→ FAC		

Evaluation de polynômes :

A l'entrée de ces routines, X pointe une table dans laquelle le premier octet donne le degré du polynôme et les suivants, par groupes de quatre, les coefficients.

POLY est un polynôme de type :

$C_0 + C_1X + C_2X^2 + C_3X^3 + ... + C_nX^N$

POLYI est un polynôme impair de type :

$C_1X + C_3X^3 + ... + C_{2n+1}X^(2N + 1)$
$2435 POLYI(FAC) → FAC
$2446 POLY (FAC) → FAC

Transferts et conversions de variables.

A l'entrée de ces routines, $6105 et le registre A doivent contenir le type de la variable à transférer (2, 4 ou 8).

$1ABF	(,X)	→	ARG		
$1CØ2	(,X)	→	FAC	$24C3	CINT
$1C36	FAC	→	(,X)	$2518	CDBL
$1C64	ARG	→	FAC	$252B	CSNG
$1C81	FAC	→	ARG		

Chercher l'erreur

Au cours du dépiautage de BASIC, il semble que pour certaines valeurs de FAC ou ARG, le programme soit débranché vers $353 lors de l'exécution de certaines routines. En y regardant de près, on trouve des syntaxes du style :

```
LDB  #$Ø6
JMP  $353     (trouvé en $1976)
```

Un EXEC &H1976 renseigne bien vite sur le rôle de cette routine $353 : c'est la routine d'affichage des messages d'erreur. Le numéro du message semble d'ailleurs être passé dans B. En essayant un mini-programme assembleur du style

```
AØØØ  C6  Ø6  LDB  #$Ø6
AØØ2  7E  Ø353  JMP  $353
```

en RAM, ce qui permet de changer le contenu chargé dans B (à l'adresse $AØØ1), on vérifie bien vite que les valeurs passées à B sont bien celles citées dans le manuel de référence.

Il paraît alors assez naturel de balayer une fois de plus les 16 K de BASIC à la recherche des appels aux routines d'erreur.

Et voilà le travail...

Appels de $Ø353	Valeur de B	Type de l'erreur
$Ø572	11	CN
$Ø658	Ø3	RG
$Ø7F6	Ø2	SN
$ØB8F	Ø5 et Ø9	FC ou BS
$ØC62	ØA	DD
$ØCDB	1Ø	ST
$ØDD4	ØF	LS
$1622	Ø1	NF
$16D8	18	WH
$17AB	14	RE
$17E8	OC	ID
$1979	Ø6	OV
$1B26	ØB	/Ø
$24BF	ØD	TM
$2786	36	IE
$2C66	37	FD
$2D3Ø	3D	PP[1]
$2EØB	32	NU
$2E64	33	FM
$2E77	3C	DU
$2F88	35	IO
$39A4	3B	IU

En désassemblant un peu autour de $ØB8F, on trouve que l'affichage du message FC ERROR est provoqué par un sous-programme qui commence en $B8C, programme qui est lui-même appelé en :

1.L'étude des environs de $2D3Ø est particulièrement riche d'enseignements.

$CD2
$EB4
$12F3
$142Ø
$17A2
$19E6
$3314
$38ØØ
$3BØD
$3F5E

Le moins que l'on puisse dire est que ceci donne une idée assez précise des différents éléments de BASIC.

De mieux en mieux...

A partir de l'adresse &HØ26B on trouve une table qui donne sur deux octets les points d'entrée des programmes binaires correspondant aux 57 premiers mots du BASIC listés p.54 (END, FOR...). On trouve une table semblable à partir de l'adresse &HØØ2Ø pour les mots 86 et suivants (SGN, INT...).

En regardant le tableau p.45-46, on peut ainsi trouver que END est équivalent à EXEC &H053B ou que RUN peut s'écrire EXEC &H05F2. Comme me l'a montré Serge Hocquengeim, du CNAM, il est bien plus savoureux de taper :

```
EXEC &HØF78 "TARTEMOLLE"
TARTEMOLLE
```

que de taper :

```
PRINT "TARTEMOLLE"
TARTEMOLLE
OK
```

Le fin limier constatant que le truc « EXEC 1216 », décrit en note de bas de p.60, démarre quelques octets après le début de NEW n'aura guère de difficultés à en expliquer le fonctionnement.

Postlude

A ce point, je dois faire une confidence au lecteur.

Le seul outil d'analyse que j'ai donné jusqu'ici est le programme « PEEK mémoire ». Il est théoriquement suffisant. Pourtant, je dispose de quelques gadgets supplémentaires, dont - un programme moniteur qui permet d'examiner et de modifier le contenu d'une mémoire en mode HEXA ou ASCII et en mode mnémonique (pour l'examen seulement, hélas !...). Ce programme fait donc tout seul le travail qui consiste à traduire les séquences d'octets en mnémoniques. C'est ce que l'on appelle un désassembleur. Je ne puis en donner le listing ici pour deux raisons : tout d'abord, il est trop long, ensuite, je souhaite évidemment faire fortune en le vendant !

Annexes

Les routines moniteur

PUTCH $E8Ø3
B contient le code vidéotex à interpréter ($Ø - $FF)...

GETSC $E824
A contient la ligne (Ø - 24). X contient la colonne (1 - 4Ø). GETSC renvoie le code caractère dans B. Si le caractère est inconnu, GETSC renvoie Ø dans B. Si le caractère est accentué, il faut appeler GETSC trois fois.

INTISC $E8ØØ
Rétablit la fenêtre texte à sa taille maximum, en mode défilement.

CHPLOT $E833
X contient la colonne (1 - 4Ø), Y contient la ligne (Ø - 24). Le registre CHDRAW ($6Ø41) contient le code du caractère à afficher. Le registre COULEUR ($6Ø3B) contient la couleur du caractère à afficher ainsi que la couleur du fond sur lequel le caractère est affiché.

PLOTXY $E8ØF
Si CHDRAW n'est pas nul, cette routine passe la main à CHPLOT. Si CHDRAW contient Ø, cette routine affiche le point de coordonnées X (Ø - 319) et Y (Ø - 199). Les couleurs d'affichage sont déterminées par le contenu de l'octet FORME ($6Ø38).

Points		Fonds
0	noir	– 1
1	rouge	– 2
2	vert	– 3
3	jaune	– 4
4	bleu	– 5
5	magenta	– 6
6	cyan	– 7
7	blanc	– 8

Sur le TO7-70, les codes de couleur sont compris entre – 16 et + 15.

GETPT $E821
On passe à cette routine les coordonnées d'un point par l'intermédiaire de X (Ø - 319) et de Y (Ø - 199). Cette routine renvoie dans B la couleur du point. Le résultat est négatif si le point appartient au fond.

DRAWXY $E80C
Cette routine dessine un vecteur depuis le dernier point tracé (PLOTXY) ou l'extrémité du dernier vecteur tracé (DRAWXY) jusqu'au point dont les coordonnées sont passées dans X et Y. Ce vecteur est un vecteur graphique si CHDRAW est nul et un vecteur de caractères si CHDRAW contient un code affichable. La couleur est déterminée par le contenu de FORME pour les tracés graphiques et par le contenu de COULEUR pour les tracés alphanumériques. Les coordonnées du point de départ du vecteur suivant se trouvent dans les registres 16 bits PLOTX ($603D - $603F) et PLOTY ($603F - $6040).

GETCH $E806
Cette routine renvoie dans l'accumulateur B le code ASCII de la dernière touche enfoncée. Si aucune touche n'a été enfoncée, B contient Ø.

KEYTST $E809
Cette routine permet de savoir où une touche est enfoncée lors de l'appel. La routine renvoie l'état du clavier dans le flag C du registre d'état du 6809. Si C = Ø, aucune touche n'est enfoncée. Si C = 1 , une touche est pressée.

K7CONT $E815
Cette routine commande toute la gestion du lecteur-enregistreur de programmes selon le mot de commande présent dans le registre

K7.OPC ($6Ø29) lors de l'appel. On peut lire un mot d'état après exécution de cette routine dans K7.STA ($6Ø2A). Si une erreur survient lors de l'exécution de cette routine, le flag C est mis à 1.

K7.OPC : $1 Ouverture en lecture
$2 Lecture d'un caractère (ce caractère est retourné dans B)
$4 Ouverture en écriture
$8 Ecriture sur cassette d'un caractère (le code du caractère est dans B)
$1Ø CLOSE (fermeture)

K7.STA : $1 Ouverture en lecture
$4 Ouvert en écriture
$1Ø Fermé
$8Ø Non prêt

Après chaque utilisation de cette routine, on testera C. Si C = Ø, tout s'est bien passé; si C = 1, on pourra lire K7.STA afin de connaître l'état du lecteur-enregistreur de programmes.

ASCONT $E812
Cette routine gère le contrôleur de communications selon le mot de commande présent dans le registre RS.OPC ($6Ø2B) lors de l'appel. Après exécution de cette routine, le flag C du 6809 est mis à Ø si tout s'est bien passé; il est mis à 1 si une erreur s'est produite. On peut lire un mot d'état après l'exécution de cette routine dans RS.STA ($6Ø2C).

RS.OPC : $1 Ouverture du RS 232 en lecture écriture
$2 Lecture d'un caractère sur le RS 232 (le caractère est retourné dans B)
$4 Ouverture du RS 232 en écriture seule
$8 Emission du caractère présent dans B
$1Ø Fermeture
$2Ø Copie de l'écran graphique sur l'imprimante thermique
$4Ø Ouverture en mode parallèle

RS.STA : $1 Ouvert en entrée-sortie série
$4 Ouvert en sortie série
$1Ø Fermé

$40 Ouvert en parallèle
$80 Non prêt

En mode série, le format du caractère émis est géré par la valeur du registre NOMBRE ($6046).
NOMBRE = $80 8 bits
NOMBRE = $40 7 bits

La vitesse de transmission est gérée par la valeur du registre BAUDS ($6044).

Bauds	Vitesse
1 130	110 bauds
411	300 bauds
202	600 bauds
98	1 200 bauds
46	2 400 bauds
21	4 800 bauds
7	9 600 bauds

MUSIC $E81E
Cette routine joue la note dont le code est passé dans l'accumulateur B en fonction de l'état des registres suivants :

	Adresse	Valeur
OCTAVE	($6036 - $6037)	1 à 16
DURÉE	($6033 - $6034)	1 à 96
TIMBRE	($6035)	0 à 255
TEMPO	($6031 - $6032)	1 à 255

Les codes passés par B vont de $30 pour le *DO*, $31 pour le *DO #*, jusqu'à $3C pour le *DO* suivant.

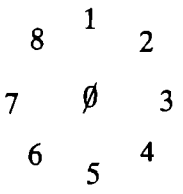
DO	$30
DO #	$31
RÉ	$32
RÉ #	$33
MI	$34
FA	$35
FA #	$36
SOL	$37
SOL #	$38

LA $39
LA # $3A
SI $3B
DO $3C

LPINT $E81B
Cette routine renvoie C = Ø si le bouton du crayon optique n'est pas enfoncé, et C = 1 s'il l'est.

GETLP $E818
Cette routine renvoie la position du crayon optique dans les registres X et Y. Si le crayon optique est hors écran ou si la lecture est impossible, X et Y contiennent – 1 ($FFF).

JOYSTK $F827
Cette routine renvoie l'état des manettes de jeu. Le numéro de la manette (Ø ou 1) est passé dans A. Le flag C donne l'état du bouton (Ø s'il n'est pas enfoncé) et la position du STICK est retournée dans B.



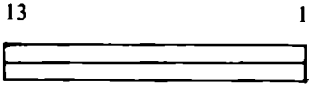
E8ØØ INITSC
E8Ø3 PUTCH
E8Ø6 GETCH
E8Ø9 KEYTST
E8ØC DRAWXY
E8ØF PLOTXY
E812 RSCONT
E815 K7CONT
E818 GETLP
E81B LPINT
E81E MUSIC
E821 GETPT
E824 GETSC
E827 JOYSTK
E82A DKCON
E82D MENU
E83Ø KBINT
E833 CHPLOT

Connecteurs

Connecteur extension mémoire

Dessus : A

Dessous : B

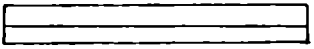


	A	B
1	+12V	+5V
2	AØ	D7
3	A1	D6
4	A2	D5
5	A3	D4
6	A4	D3
7	A5	D2
8	A6	D1
9	R/W	DØ
1Ø	CLOCK	OE
11	RAS	CAS
12	GND	GND
13	GND	-5V

Connecteurs extensions

Dessus : A

Dessous : B

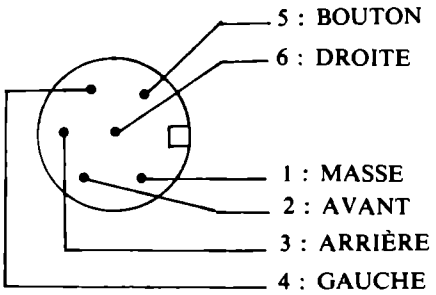


	A	B
1	GND	+12V
2	NC	NC
3	E7C	SON
4	CSE	R/W
5	A11	E
6	A1Ø	D7
7	A9	D6
8	A8	D5
9	A7	D4

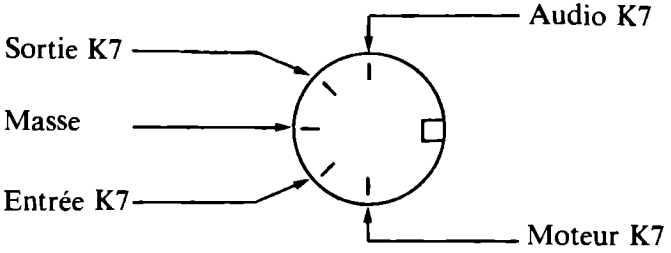
1Ø	A6	D3
11	A5	D2
12	A4	D1
13	A3	DØ
14	A2	$\overline{RST}$
15	A1	$\overline{FIRQ}$
16	AØ	$\overline{NMI}$
17	NC	$\overline{IRQ}$
18	NC	VIDEO
19	+5V	-5V

SON : entrée son
E7C : sélection de E7CX
CSE : sélection de EØØØ-EFFF
VIDEO : broche 19 de la prise Péritel
E : validation de l'extension
NC : non connecté

Prise JEUX



Prise LEP



En faisant attention de ne pas tirer trop de courant, on peut utiliser la broche 1 pour commander toutes sortes de dispositifs à l'aide des instructions MOTORON et MOTOROFF.

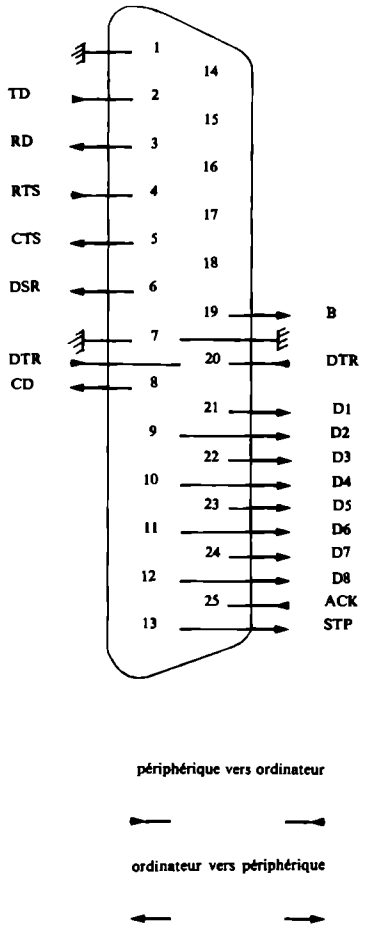
Prise RS-232 : interface parallèle

Série

1 Masse
2 Transmission donnée
3 Réception donnée
4 Prêt pour émission
5 R à Z pour émettre
6 Données prêtes
7 Masse
8 Détection de porteuse
2Ø Terminal prêt

Parallèle

19 Sortie « périphérique occupé »
2Ø Terminal prêt
21 Donnée 1
9 Donnée 2
22 Donnée 3
1Ø Donnée 4
23 Donnée 5
11 Donnée 6
24 Donnée 7
12 Donnée 8
25 Accusé de réception
13 Contrôle de validation (strobe)

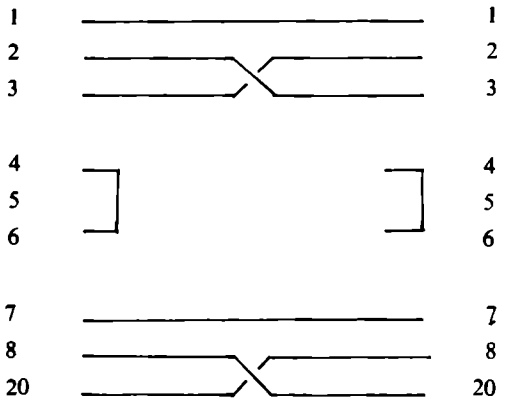


Brochage de la prise Cannon (côté périphérique)

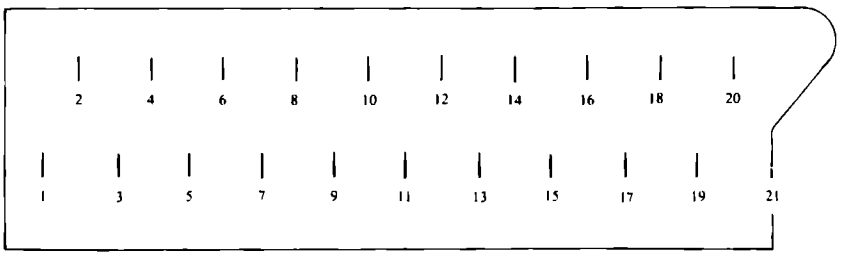
Si l'on ne désire pas gérer les signaux 19 et 2Ø de l'interface parallèle, on peut réunir ces deux broches. On évitera ainsi des I/O ERROR.

De même, sur l'interface série, on peut réunir les broches 4 et 5 d'une part et 6, 8 et 2Ø d'autre part si l'on ne désire pas gérer ces signaux.

Liaison entre deux TO7 via RS-232



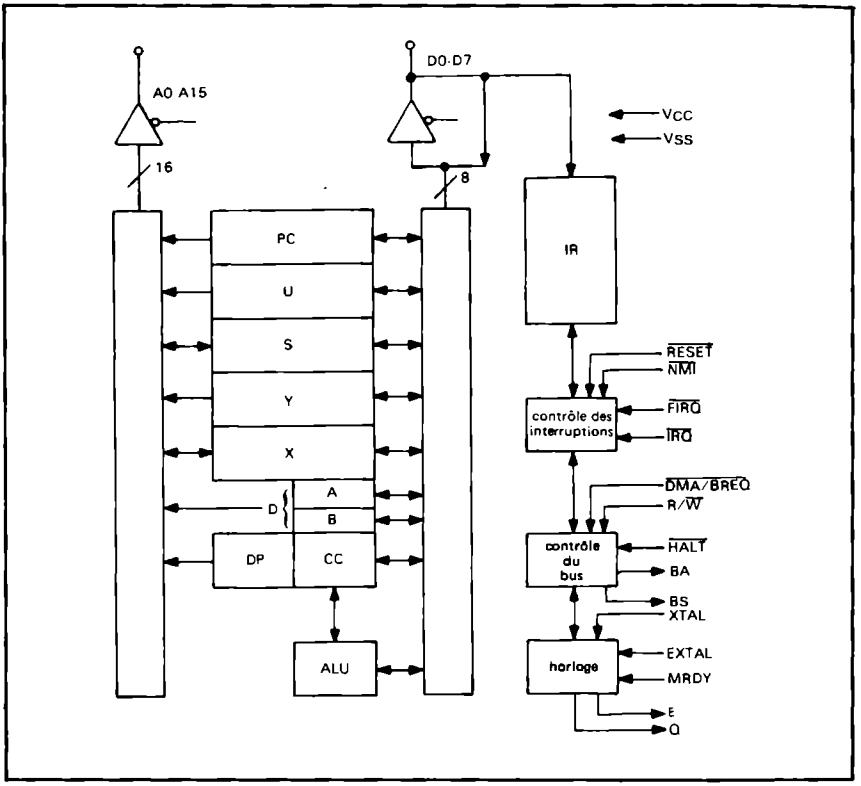
Prise Péritel (SCART)



1 Sortie audio B
2 Entrée audio B
3 Sortie audio A
4 Masse audio
5 Masse BLEU
6 Entrée audio A
7 BLEU
8 Commutation lente (appliquer +12 V pour valider les entrées de la prise Péritel)
9 Masse VERT
1Ø Intercommunication
11 VERT
12 Intercommunication
13 Masse ROUGE
14 Masse intercommunication
15 ROUGE
16 Commutation rapide (appliquer +3 V pour valider les entrées de la prise Péritel)
17 Vidéo
18 Masse commutation rapide
19 Sortie vidéo
2Ø Entrée vidéo ou synchro
21 Masse

Le 6809

Le 6809 est le microprocesseur qui équipe le TO7. C'est un circuit très puissant qui possède un peu moins d'une centaine d'instructions de base. Ces instructions possèdent en général une grande diversité de syntaxe.



BROCHAGE

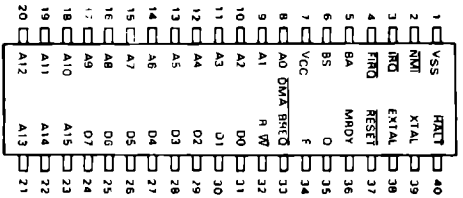
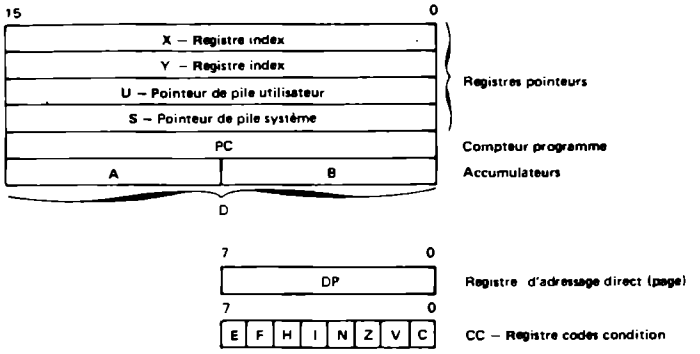


FIGURE 5 – REGISTRES PROGRAMMABLES DU MICROPROCESSEUR



REGISTRES PROGRAMMABLES

Comme indiqué ci-dessus, le microprocesseur 6809 comporte trois registres supplémentaires par rapport au 6800. Ces registres sont les suivants : un registre de page directe (DP), un registre pointeur de pile utilisateur (U) et un second registre index (Y).

ACCUMULATEURS (A, B, D)

Les registres A et B sont des accumulateurs universels utilisés pour les calculs arithmétiques et les manipulations de données.
Certaines instructions concatènent les registres A et B pour former un seul accumulateur 16 bits. Le registre A constitue l'octet de poids fort de cet accumulateur référencé registre D.

REGISTRE PAGE DIRECTE (DP)

Le registre de page directe du circuit 6809 est utilisé pour étendre les possibilités d'adressage en mode direct. Le contenu de ce registre apparaît aux sorties d'adresse de poids forts (A8-A15) pendant l'exécution d'une instruction d'adressage direct. Ce registre permet d'utiliser le mode d'adressage direct, sous le contrôle du programme, dans tout l'espace d'adressage. Pour permettre la compatibilité avec la famille 6800, tous les bits de ce registre sont mis à zéro à l'initialisation du processeur.

REGISTRES INDEX (X, Y)

Les registres d'index sont utilisés pour les modes d'adressage indexé. Lors des calculs d'adresse effective, les 16 bits de ce registre sont utilisés. Les adresses contenues dans ces registres peuvent servir comme

pointeur de données et être modifiées par une constante optionnelle ou par une valeur de déplacement. Lors de rangement de données sous forme de table, dans certains modes d'adressage indexé, le contenu des registres d'index est incrémenté ou décrémenté pour pointer sur l'élément suivant. Les quatre registres (X, Y, U, S) peuvent être utilisés comme des registres d'index.

POINTEURS DE PILE (U,S)

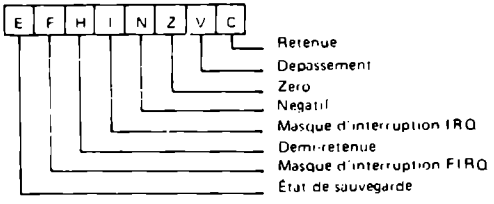
Le pointeur de pile (S) est utilisé automatiquement par le processeur pour mémoriser les états de la machine pendant l'exécution de sous-programmes et interruptions. Les pointeurs du 6809 pointent le haut de la pile, à l'opposé du pointeur du 6800, qui pointait l'emplacement libre suivant sur la pile. Le pointeur de pile utilisateur (U) est commandé par le programmeur exclusivement, permettant ainsi le passage de paramètres de et vers des sous-programmes avec facilité. Les pointeurs de pile U et S ont les mêmes possibilités que les registres X et Y pour les modes d'adressage indexé et pour les instructions d'empilement/dépilement. Le micro processeur 6809 peut être utilisé comme processeur avec gestion de pile, autorisant ainsi l'utilisation de langage de haut niveau et des méthodes de programmation modulaire.

COMPTEUR PROGRAMME (PC)

Le compteur programme est utilisé par le processeur pour pointer l'adresse de l'instruction suivante devant être exécutée. L'adressage relatif permet au compteur programme d'être utilisé comme un registre index dans certains cas.

REGISTRE DE CODES CONDITION (CC)

Le registre de codes condition définit l'état du processeur à tout instant.



DESCRIPTION DU REGISTRE CODES CONDITION (CC)

BIT 0 (C)
Le bit 0 est l'indicateur de retenue, il indique généralement la retenue lors d'une opération de l'unité arithmétique et logique. C est aussi utilisé pour représenter la retenue lors d'instructions correspondant à une soustraction (CMP, NEG, SUB, SBC). Dans ce cas cet indicateur est le complément de la retenue lors d'une opération de l'unité logique et arithmétique.

BIT 1 (V)
Le bit 1 est l'indicateur de débordement ; il est mis à un s'il y a débordement, en complément à deux signé après une opération arithmétique. Le débordement est détecté lors d'une opération dans l'unité logique et arithmétique quand la retenue du MSB ne correspond pas à la retenue du MSB-1.

BIT 2 (Z)
Le bit 2 est le bit indicateur de zéro, il est mis à un si le résultat de l'opération précédente est nul.

BIT 3 (N)
Le bit 3 indique un résultat négatif, il contient exactement la valeur du bit de poids fort de l'octet résultant de l'opération précédente. Un résultat négatif, en complément à deux, positionne N à 1.

BIT 4 (I)
Le bit 4 est le bit masque des interruptions IRQ. Ce bit mis à un, le processeur ne prendra pas en compte les interruptions arrivant sur la ligne IRQ. NMI, FIRQ, IRQ, RESET et SWI positionnent toutes 1 à un. SWI 2 et SWI 3 n'affectent pas 1.

BIT 5 (H)
Le bit 5 est le bit de demi-retenue ; il est utilisé pour indiquer une retenue du bit 3 dans l'ALU comme résultat d'une addition 8 bits seulement (ADC ou ADD). Ce bit est utilisé dans une instruction DAA pour réaliser une opération d'ajustement décimal. L'état de cet indicateur est indéfini dans toutes les instructions de soustraction ou équivalentes.

BIT 6 (F)
Le bit 6 est le bit masque des interruptions rapides FIRQ. Le processeur ne prendra pas en compte les interruptions de la ligne FIRQ, lorsque ce bit est à un. NMI, FIRQ, SWI, et RESET positionnent toutes F à un. F n'est pas affecté par IRQ, SWI 2 et SWI 3.

BIT 7 (E)
Le bit E est le bit indicateur de l'état de sauvegarde ; mis à un, il indique que l'état complet de la machine (tous les registres) est empilé, à la place de l'état précédent (PC et CC). Le bit E du registre CC empilé est utilisé sur un retour d'interruption (RTI) pour déterminer l'étendue du dépilement. Par conséquent, le bit E courant laissé dans le registre CC représente l'action précédente.

MODES D'ADRESSAGE

Les instructions de base de tout ordinateur sont particulièrement améliorées par la présence de modes d'adressage puissants. Le jeu de modes d'adressage disponible du 6809 est actuellement le plus puissant des microprocesseurs existants.
Par exemple, le 6809 possède 59 instructions de base, mais il admet 1464 possibilités différentes d'instructions et de modes d'adressage. Les nouveaux modes d'adressage permettent les techniques de programmation modernes. Les modes d'adressage suivants sont disponibles dans le 6809 :

Inhérent ou implicite (Inclut les accumulateurs)
Immédiat
Etendu
Etendu indirect
Direct
Registre
Indexé
  Déplacement nul
  Déplacement constant
  Déplacement accumulateur
  Auto incrémentation/décrémentation
  Indexé indirect
Relatif
  Branchement relatif long/court
  Adressage relatif compteur programme

INHÉRENT (INCLUT LES ACCUMULATEURS)
Dans ce mode d'adressage, le code opération de l'instruction contient toute l'information adresse nécessaire. Des exemples d'adressage inhérent sont : ABX, DAA, SWI, ASRA, CLRB etc.

ADRESSAGE IMMÉDIAT
En adressage immédiat, l'adresse effective des données se trouve à l'emplacement suivant immédiatement le code opération ; les données à utiliser comme adresse dans l'instruction du 6809 utilisent deux valeurs immédiates 8 et 16 bits en fonction de la taille de l'opérande spécifié par le code opération. Des exemples d'instructions utilisant l'adressage immédiat sont :

```
LDA     #$20
LDX     #$F000
LDY     #ASTER
```

Note : # signifie adressage immédiat, $ signifie valeur hexadécimale.

ADRESSAGE ÉTENDU
En adressage étendu, le contenu des deux octets suivant immédiatement le code opération spécifie complètement l'adresse 16 bits effective utilisée par l'instruction.
Il est à noter que l'adresse générée par une instruction étendue définit une adresse absolue et n'est pas translatable. Les exemples d'adressage étendu incluent :

```
LDA     ASTER
STX     OBEL
LDD     $2000
```

ÉTENDU INDIRECT
Comme cas spécial d'adressage indexé (exposé ci-dessous), un niveau d'indirection peut être ajouté à l'adressage étendu. En mode étendu indirect, les deux octets suivant le post octet d'une instruction indexée contiennent l'adresse de l'adresse des données.

```
LDA     [ASTER]
LDX     [$ FFFE]
STU     [OBEL]
```

ADRESSAGE DIRECT

L'adressage direct est similaire à l'adressage étendu excepté qu'un octet d'adresse seulement suit le code opération. Cet octet spécifie les 8 bits de poids faible de l'adresse à utiliser. Les 8 bits d'adresse de poids fort sont fournis par le registre page directe. Un octet d'adresse étant seulement nécessaire en adressage direct, ce mode nécessite moins de mémoire et s'exécute plus rapidement qu'en adressage étendu. Bien entendu, seuls 256 emplacements (une page) peuvent être définis sans avoir à repositionner le contenu du registre DP. Le registre DP étant mis à $ 00 à l'initialisation, l'adressage direct sur le 6809 est compatible avec l'adressage direct du 6800. L'indirection n'est pas permise en adressage direct.

Voici quelques exemples d'adressage direct :

```
LDA     $30
SETDP   $10 (directive d'assemblage)
LDB     $1030
LDD     <ASTER
```

Note : < est une directive d'assemblage qui force l'adressage direct.

ADRESSAGE PAR REGISTRE

Certains codes opération sont suivis d'un octet qui définit un registre ou un jeu de registres devant être utilisés par l'instruction, cet octet est appelé POST OCTET.

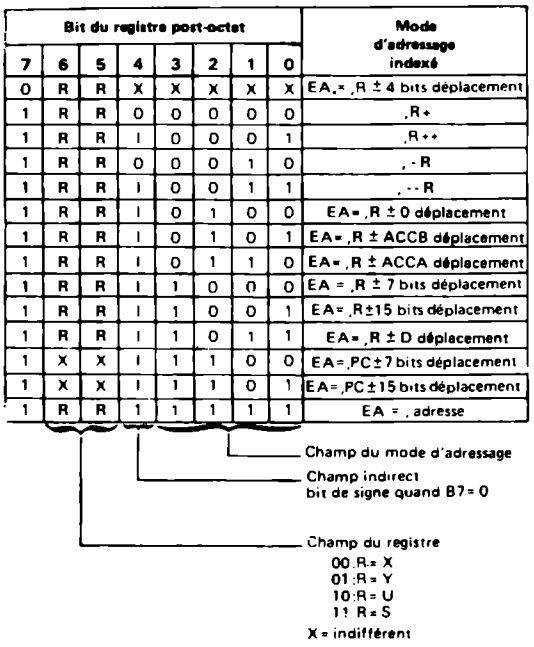
Quelques exemples d'adressage registre sont :

```
TFR     X,Y     Transfert de X dans Y
EXG     A,B,    Échange A et B
PSHS    A,B,X,Y Transfert dans S  Y, X, B puis A
PULU    X, Y, D Transfert depuis U  D, X, puis Y
```

ADRESSAGE INDEXÉ

Dans tout adressage indexé, un des registres pointeur (X, Y, U, S et parfois PC) est utilisé dans le calcul de l'adresse effective de l'opérande devant être utilisée par l'instruction. Cinq types d'indexation de base sont disponibles et sont exposés ci-dessous. Le post octet d'une instruction indexée spécifie le type de base et le choix du mode d'adressage ainsi que le registre pointeur devant être utilisé. Le tableau ci-dessous montre les formats autorisés pour le post octet. Le tableau "MODES d'Adressage INDEXÉ" donne la forme assembleur et le nombre de cycles et d'octets additionnés aux valeurs de base d'adressage indexé pour chaque variante.

Signification des bits du registre post-octet dans l'adressage indexé



Bit du registre post-octet								Mode d'adressage indexé
7	6	5	4	3	2	1	0	
0	R	R	X	X	X	X	X	EA = ,R ± 4 bits déplacement
1	R	R	0	0	0	0	0	,R+
1	R	R	1	0	0	0	1	,R++
1	R	R	0	0	0	1	0	,-R
1	R	R	1	0	0	1	1	,--R
1	R	R	1	0	1	0	0	EA = ,R ± 0 déplacement
1	R	R	1	0	1	0	1	EA = ,R ± ACCB déplacement
1	R	R	1	0	1	1	0	EA = ,R ± ACCA déplacement
1	R	R	1	1	0	0	0	EA = ,R ± 7 bits déplacement
1	R	R	1	1	0	0	1	EA = ,R ± 15 bits déplacement
1	R	R	1	1	0	1	1	EA = ,R ± D déplacement
1	X	X	1	1	1	0	0	EA = ,PC ± 7 bits déplacement
1	X	X	1	1	1	0	1	EA = ,PC ± 15 bits déplacement
1	R	R	1	1	1	1	1	EA = , adresse

Indexé — Déplacement zéro. Dans ce mode, le registre pointeur sélectionné contient l'adresse effective des données devant être utilisées par l'instruction. Ce mode est le mode indexé le plus rapide. Exemples :

LDD 0,X
LDA 0,S

Indexé — Déplacement constant. Dans ce mode, un déplacement en complément à deux et le contenu d'un des registres pointeurs sont additionnés pour former l'adresse effective de l'opérande. Le contenu initial du registre pointeur n'est pas changé par l'addition. Trois valeurs de déplacement sont disponibles.

± 4-bit(− 16 à + 15)
± 7-bit(− 128 à + 127)
± 15-bit(− 32 768 à + 32 767)

Le déplacement 5 bits en complément à deux est compris dans le post octet et donc optimise l'utilisation des octets et des cycles. Le déplacement 8 bits en complément à deux est contenu dans un seul octet suivant le post octet. Le déplacement 16 bits en complément à

deux se trouve dans les deux octets suivant le post octet. Dans la plupart des cas, le programmateur n'a pas à connaître la valeur de ce déplacement puisque l'assembleur sélectionne automatiquement la valeur d'option.
Exemples d'indexation avec déplacement constant :

```
LDA     23,X
LDX     -2,S
LDY     300,X
LDU     ASTER,Y
```

Mode d'adressage indexé : non indirect

Type	Formes	Non Indirect			
		Syntaxe assembleur	Post-octet code OP	+	~ #
Déplacement constant à partir de R (signé)	pas de déplacement	R	1RR00100	0	0
	déplacement 5 bits	n R	0RRnnnnn	1	0
	déplacement 8 bits	n R	1RR01000	1	1
	déplacement 16 bits	n R	1RR01001	4	2
Accumulateur utilisé comme déplacement pour le Registre R (déplacement signé)	registre de déplac. A	A R	1RR00110	1	0
	registre de déplac. B	B R	1RR00101	1	0
	registre de déplac. D	D R	1RR01011	4	0
Auto incrémentation/décrémentation du registre R	incrémenté par 1	R+	1RR00000	2	0
	incrémenté par 2	R++	1RR00001	3	0
	décrémenté par 1	-R	1RR00010	2	0
	décrémenté par 2	--R	1RR00011	3	0
Déplacement constant à partir de PC	déplacement 8 bits	n PCR	1XX01100	1	1
	déplacement 16 bits	n PCR	1XX01101	5	2
Indirect étendu	adresses 16 bits	—	—	--	

R = X, Y, U ou S — X : 00 Y : 01
X = indifférent — U : 10 S : 11

+ et ~ # indiquent le nombre de cycles et d'octets additionnels pour un état particulier.

Mode d'adressage indexé : indirect

Type	Formes	Indirect			
		Syntaxe assembleur	Post-octet code OP	~	~ #
Déplacement constant à partir de R (signé)	pas de déplacement	[R]	1RR10100	3	0
	déplacement 5 bits	par défaut = 8 bits			
	déplacement 8 bits	[n R]	1RR11000	4	1
	déplacement 16 bits	[n R]	1RR11001	7	2
Accumulateur utilisé comme déplacement pour le Registre R (déplacement signé)	registre de déplac. A	[A R]	1RR10110	4	0
	registre de déplac. B	[B R]	1RR10101	4	0
	registre de déplac. D	[D R]	1RR11011	7	0
Auto incrémentation/décrémentation du registre R	incrémenté par 1	impossible			
	incrémenté par 2	[R++]	1RR10001	6	0
	décrémenté par 1	impossible			
	décrémenté par 2	[--R]	1RR10011	6	0
Déplacement constant à partir de PC	déplacement 8 bits	[n PCR]	1XX11100	4	1
	déplacement 16 bits	[n PCR]	1XX11101	8	2
Indirect étendu	adresses 16 bits	[n]	10011111	5	2

R = X, Y, U ou S — X : 00 Y : 01
X = indifférent — U : 10 S : 11

+ et ~ # indiquent le nombre de cycles et d'octets additionnels pour un état particulier.

MODES D'ADRESSAGE INDEXÉ

Indexé — Déplacement accumulateur. Ce mode est semblable au mode indexé à déplacement constant, excepté que la valeur en complément à deux dans un des accumulateurs (A, B ou D) et le contenu de l'un des registres pointeurs sont ajoutés pour former l'adresse effective de l'opérande. Le contenu du registre pointeur et de l'accumulateur demeure inchangé par l'addition. Le post octet spécifie l'accumulateur à utiliser comme déplacement et aucun octet supplémentaire n'est nécessaire. L'avantage d'un déplacement accumulateur réside dans le fait que la valeur du déplacement peut être calculée par programme en cours d'exécution.

Exemples :

```
LDA     B, Y
LDX     D, Y
LEAX    B, X
```

Indexé — Auto Incrémentation/Décrémentation. En mode auto incrémentation, le registre pointeur contient l'adresse de l'opérande. Ainsi, après avoir été utilisé le registre pointeur est incrémenté de un ou deux. Ce mode d'adressage est très utile lors de l'utilisation de tables, de déplacement de données, ou pour la création de piles logicielles. En auto décrémentation le registre pointeur est décrémenté avant d'être utilisé comme adresse des données. L'utilisation en auto décrémentation est similaire à celle en auto incrémentation, mais les tables sont scrutées les adresses élevées vers les adresses faibles. La valeur d'incrément/décrément peut être égale à un ou deux pour permettre d'accéder à des tables de données 8 ou 16 bits, elle est sélectionnée par le programmeur. L'aspect pré-décrément, post-incrément permet à ces modes d'être utilisés pour créer des piles logicielles supplémentaires qui se comportent de manière identique aux piles U et S.

Voici quelques exemples de modes d'adressage auto incrément/décrément :

```
LDA     ,X+
STD     ,Y++
LDB     ,-Y
LDX     ,--X
```

INDEXE INDIRECT

Tous les modes indexé indirect sont inclus à l'exception d'incrémentation/décrémentation par un, ou déplacement de ±4 bits et peu-

vent avoir un niveau d'indirection supplémentaire spécifié. En adressage indirect, l'adresse effective est contenue à l'emplacement spécifié par le contenu du registre index, additionné d'un quelconque déplacement. Dans l'exemple ci-dessous, l'accumulateur A est chargé indirectement en utilisant une adresse effective calculée à partir du registre index et d'un déplacement.

```
Avant exécution :
A = x x (indifférent)
X = $ . F000
$0100 LDA [10,X] l'EA est alors $F010
$F010 $F1        F150 est alors la nouvelle
$F011 $50        adresse effective
$F150 $AA
Après exécution :
A = $AA Donnée chargée
```

Note : EA = adresse effective.
Tous les modes indexé indirect sont inclus à l'exception de ceux qui sont sans signification (exemple : auto incrément/décrément par 1 indirect). Quelques exemples de mode indexé indirect sont :

```
LDA    [,X]
LDD    [10,X]
LDA    [B,Y]
LDD    [,X+ +]
```

ADRESSAGE RELATIF

Le(s) octet(s) suivant(s) le code opération de branchement est (sont) traité(s) comme un déplacement signé qui est additionné au compteur programme.
Si la condition de branchement est vraie, alors l'adresse calculée (PC + déplacement signé) est chargée dans le compteur programme. L'exécution du programme se poursuit jusqu'au nouvel emplacement comme indiqué par le PC, les modes d'adressage relatif court (1 octet de déplacement) et long (déplacement de deux octets) sont disponibles. Tout emplacement mémoire peut être atteint en mode d'adressage relatif long, l'adresse effective étant interprétée modulo 2^{16}. Quelques exemples d'adressage relatif sont :

```
        BEQ     ASTER    (court)
        BGT     OBEL     (court)
ASTER   LBEQ    BAMBI    (long)
OBEL    LBGT    BUNNY    (long)
```

```
BAMBI       NOP
BUNNY       NOP
```

Le compteur programme peut être utilisé comme registre pointeur avec des déplacements signés de 8 ou 16 bits. Comme en adressage relatif le déplacement est additionné au PC en cours pour former l'adresse effective. L'adresse effective est alors utilisée comme adresse opérande ou adresse données. L'adressage relatif par compteur programme est utilisé pour écrire des programmes translatables. Les tables relatives à un programme particulier gardent la même liaison après translation du programme, si celles-ci sont référenciées en relatif par rapport au compteur programme.

Exemples :

```
LDA     BUNNY, PCR
LEAX    TABLE, PCR
```

Le mode compteur programme relatif étant un type d'indexation, un niveau supplémentaire d'indirection est utilisable.

```
LDA     [ASTER,PCR]
LDU     [OBEL,PCR]
```

JEU D'INSTRUCTIONS DU 6809

Le jeu d'instruction du 6809 est comparable à celui du 6800 et compatible ascendant au niveau du code source. Le nombre de codes opération a été réduit de 72 à 59, mais grâce à son architecture améliorée et modes d'adressage supplémentaires, le nombre de codes opération disponibles (avec les différents modes d'adressage) est passé de 197 à 1 464.

Certaines instructions et certains modes d'adressage sont décrits en détail ci-dessous :

PSHU/PSHS

Ces instructions ont la propriété d'empiler tout(s) registre(s) du MPU soit sur la pile matériel (S) soit sur la pile utilisateur (U) en une seule instruction.

PULU/PULS

Les instructions de dépilement ont la même propriété que les instructions d'empilement, dans l'ordre inverse. L'octet immédiat suivant

le code opération des instructions d'empilement ou de dépilement détermine quel ou quels registres doivent être empilés ou dépilés. La séquence effective d'empilement/dépilement est fixée ; chaque bit détermine un registre unique à empiler/dépiler comme indiqué.

POST OCTET D'EMPILEMENT/DÉPILEMENT

← ordre d'empilement ordre de dépilement →

PC	U	Y	X	DP	B	A	CC	PSHS/PULS

FFFF... ←adresse mémoire croissante... 0000

PC	S	Y	X	OP	B	A	CC	PSHU/PULU

TFR/EXG

Dans le 6809 chaque registre peut être transféré ou échangé avec un autre registre de même format, c'est-à-dire 8 bits à 8 bits ou 16 bits à 16 bits. Les bits 4-7 du post octet définissent le registre source, tandis que les bits 0-3 représentent le registre destination.
Ceci se représente comme suit :

0000 – D	0101 – PC
0001 – X	1000 – A
0010 – Y	1001 – B
0011 – U	1010 – CC
0100 – S	1011 – DP

Note : Toutes les autres combinaisons sont indéfinies et non valables.

Chargement d'adresse effective (LEA)

L'instruction LEA s'exécute en calculant l'adresse effective utilisée dans une instruction indexée et mémorise cette valeur d'adresse, au lieu des données de cette adresse, dans un registre pointeur. Ceci met l'ensemble des caractéristiques d'adressage interne matériel à la disposition du programmeur. Quelques-unes des implications de cette instruction sont illustrées à l'aide d'exemples.
L'instruction LEA permet aussi à l'utilisateur d'accéder à des données quel que soit l'emplacement. Par exemple :

```
        LEAX    MSG1,PCR
        LBSR    PDATA (programme d'impression message)

MSG1    FCC     /MESSAGE/
```

Cet ensemble de programme imprime "message". En écrivant MSG1, PCR, l'assembleur calcule la distance entre l'adresse présente et MSG1. Ce résultat est placé comme une constante dans l'instruction LEAX qui est indexée par la valeur du PC au moment de l'exécution. Peu importe la position du code pendant son exécution puisque le déplacement calculé depuis le PC mettra l'adresse absolue de MSG1 dans le registre pointeur X. Ce code est totalement translatable.

EXEMPLES D'UTILISATION DE L'INSTRUCTION LEA

Instruction	Opération	Commentaire
LEAX 10, X	X + 10 →X	Addition constante sur 5 bits de 10 dans X
LEAX 500, X	X + 500 →X	Addition constante sur 16 bits de 500 dans X
LEAY A, Y	Y + A →Y	Addition de l'accumulateur sur 8 bits dans Y
LEAY D, Y	Y + D →Y	Addition de l'accumulateur D sur 16 bits dans Y
LEAU -10, U	U - 10 →U	Soustraction de 10 dans U
LEAS -10, S	S - 10 →S	Réservation d'une zone dans la pile
LEAS 10, S	S + 10 →S	Remise en ordre de la pile
LEAX 5, S	S + 5 →X	Transfert aussi bien qu'addition

MUL

Multiple les nombres binaires non signés des accumulateurs A et B et place le résultat non signé dans l'accumulateur 16 bits D.

BRANCHEMENTS RELATIFS LONG ET COURT

Le 6809 a la possibilité de réaliser des branchements relatifs au compteur programme sur tout l'espace mémoire. Dans ce mode, en cas de branchement, le déplacement signé de 8 ou 16 bits est additionné à la valeur du compteur programme utilisé comme adresse effective. Ceci permet le branchement du programme n'importe où dans les 64 K d'espace mémoire. Le code translatable peut être facilement généré par l'utilisation du branchement relatif. Les deux branchements court (8 bits) et long (16 bits) sont disponibles.

TABLEAU DE JEU D'INSTRUCTIONS DU 6809

Les instructions du 6809 ont été séparées en cinq catégories différentes qui sont :

- Fonctionnement 8 bits
- Fonctionnement 16 bits
- Instructions portant sur le registre index/pointeur de pile
- Branchements relatifs (long et court)
- Instructions diverses
- Instructions sur valeur hexadécimale

INSTRUCTIONS SUR LES ACCUMULATEURS ET LA MÉMOIRE (8 BITS)

Mnemoniques	Opérations
ADCA ADCB	Addition du contenu mémoire à l'accumulateur, avec retenue
ADDA ADDB	Addition du contenu mémoire à l'accumulateur
ANDA ANDB	ET logique entre mémoire et l'accumulateur
ASL ASLA ASLB	Décalage arithmétique à gauche du contenu mémoire ou accumulateur
ASR, ASRA, ASRB	Décalage arithmétique à droite du contenu mémoire ou accumulateur
BITA, BITB	Test de bit mémoire avec l'accumulateur
CLR CLRA CLRB	Mise à zéro du contenu de l'accumulateur ou de la mémoire
CMPA, CMPB	Comparaison du contenu mémoire avec l'accumulateur
COM COMA COMB	Complément à deux de l'accumulateur ou du contenu mémoire
DAA	Ajustement décimal de l'accumulateur A
DEC DECA DECB	Décrémentation du contenu mémoire ou de l'accumulateur
EORA EORB	« OU » exclusif du contenu mémoire avec l'accumulateur
EXG R1 R2	Echange de R_1 avec R_2 (R_1, R_2 = A, B, CC, DP)
INC INCA INCB	Incrémentation du contenu mémoire ou de l'accumulateur
LDA, LDB	Chargement de l'accumulateur avec le contenu mémoire
LSL LSLA LSLB	Décalage logique à gauche du contenu mémoire ou de l'accumulateur
LSR LSRA, LSRB	Décalage logique à droite du contenu mémoire ou de l'accumulateur
MUL	Multiplication non signée (A x B → D)
NEG NEGA NEGB	Complément à un du contenu mémoire ou de l'accumulateur
ORA, ORB	« OU » logique mémoire et accumulateur
ROL ROLA ROLB	Décalage circulaire à gauche du contenu mémoire ou de l'accumulateur
ROR RORA RORB	Décalage circulaire à droite du contenu mémoire ou de l'accumulateur
SBCA SBCB	Soustraction du contenu mémoire de l'accumulateur
STA, STB	Mise en mémoire du contenu de l'accumulateur
SUBA SUBB	Soustraction du contenu mémoire de l'accumulateur
TST TSTA TSTB	Test mémoire ou accumulateur
TFR R1 R2	Transfert de R_1 à R_2 (R_1, R_2 = A, B, CC, DP)

NOTE : A, B, CC ou DP peuvent être empilés sur (ou dépilés de) chaque pile avec les instructions PSHS, PSHU, (PULS, PULU).

INSTRUCTIONS SUR LES ACCUMULATEURS ET LA MÉMOIRE (16 BITS)

Mnemoniques	Opérations
ADDD	Addition du contenu mémoire à l'accumulateur D
CMPD	Comparaison du contenu mémoire avec l'accumulateur D
EXG D R	Échange de D avec X, Y, S, U ou PC
LDD	Chargement de l'accumulateur D avec le contenu mémoire
SEX	Extension de signe de l'accumulateur B à l'accumulateur A.
STD	Mise en mémoire de l'accumulateur D.
SUBD	Soustraction du contenu mémoire de l'accumulateur D
TFR D R	Transfert de D vers X, Y, S, U ou PC
TFR R D	Transfert de X, Y, S, U ou PC vers D.

INSTRUCTIONS SUR LES REGISTRES INDEX ET LE POINTEUR DE PILE

Mnemoniques	Opérations
CMPS CMPU	Comparaison mémoire avec pointeur de pile
CMPX CMPY	Comparaison mémoire avec registre index
EXG R1 R2	Echange de D, X, Y, S, U ou PC avec D, X, Y, S, U, ou PC
LEAS LEAU	Chargement de l'adresse effective dans le pointeur de pile
LEAX LEAY	Chargement de l'adresse effective dans le registre index
LDS LDU	Chargement du pointeur de pile avec le contenu mémoire
LDX LDY	Chargement du registre index avec le contenu mémoire
PSHS	Empilement de tout (s) registre (s) (sauf S) sur la pile S
PSHU	Empilement de tout (s) registre (s) (sauf U) sur la pile U
PULS	Dépilement de tout (s) registre (s) (sauf S) de la pile S
PULU	Dépilement de tout (s) registre (s) (sauf U) de la pile U
STS STU	Mise en mémoire du pointeur de pile
STX STY	Mise en mémoire du registre index
TFR R1 R2	Transfert de D, X, Y, S, U ou PC vers D, X, Y, S, U ou PC
ABX	Addition de l'accumulateur B à X (non signé)

INSTRUCTIONS DE BRANCHEMENT

Mnemoniques	Opérations
BCC LBCC	Branchement si pas de retenue
BCS LBCS	Branchement si retenue
BEQ LBEQ	Branchement si égal
BGE LBGE	Branchement si supérieur ou egal (signé)
BGT LBGT	Branchement si supérieur (signé)
BHI LBHI	Branchement si supérieur (non signé)
BHS LBHS	Branchement si supérieur ou égal (non signé)
BLE LBLE	Branchement si inférieur ou égal (signé)
BLO LBLO	Branchement si inférieur (non signé)
BLS LBLS	Branchement si inférieur ou égal (non signé)
BLT LBLT	Branchement si inférieur (signé)
BMI LBMI	Branchement si négatif
BNE LBNE	Branchement si non égal
BPL LBPL	Branchement si positif
BRA LBRA	Branchement inconditionnel
BRN LBRN	Non branchement
BSR LBSR	Branchement à un sous programme
BVC LBVC	Branchement si pas de débordement = 0
BVS LBVS	Branchement si débordement = 1

INSTRUCTIONS SPÉCIALES

Mnemoniques	Opérations
ANDCC	« ET » logique avec le registre codes condition
CWAI	« ET » logique avec le registre codes condition puis attente d'interruption
NOP	Non operation
ORCC	« OU » logique avec le registre codes condition
JMP	Saut inconditionnel
JSR	Saut à un sous programme
RTI	Retour d'interruption
RTS	Retour de sous-programme
SWI SWI2 SWI3	Interruption programmee
SYNC	Synchronisation avec la ligne d'interruption

Code OP	Mnémonique	Mode	~	#
00	NEG	Direct	6	2
01	*			
02	*			
03	COM		6	2
04	LSR		6	2
05	*			
06	ROR		6	2
07	ASR		6	2
08	ASL/LSL		6	2
09	ROL		6	2
0A	DEC		6	2
0B	*			
0C	INC		6	2
0D	TST		6	2
0E	JMP		3	2
0F	CLR	Direct	6	2
10	Page 2	–	–	–
11	Page 3	–	–	–
12	NOP	Implicite	2	1
13	SYNC	Implicite	2	1
14	*			
15	*			
16	LBRA	Relatif	5	3
17	LBSR	Relatif	9	3
18	*			
19	DAA	Implicite	2	1
1A	ORCC	Immédiat	3	2
1B	*			
1C	ANDCC	Immédiat	3	2
1D	SEX	Implicite	2	1
1E	EXG		8	2
1F	TFR	Implicite	6	2
20	BRA	Relatif	3	2
21	BRN		3	2
22	BHI		3	2
23	BLS		3	2
24	BHS/BCC		3	2
25	BLO/BCS		3	2
26	BNE		3	2
27	BEQ		3	2
28	BVC		3	2
29	BVS		3	2
2A	BPL		3	2
2B	BMI		3	2
2C	BGE		3	2
2D	BLT		3	2
2E	BGT		3	2
2F	BLE	Relatif	3	2

Code OP	Mnémonique	Mode	~	#
30	LEAX	Indexé	4•	2•
31	LEAY		4•	2•
32	LEAS		4•	2•
33	LEAU	Indexé	4•	2•
34	PSHS	Implicite	5•	2
35	PULS		5•	2
36	PSHU		5•	2
37	PULU		5•	2
38	*			
39	RTS		5	1
3A	ABX		3	1
3B	RTI		6/15	1
3C	CWAI		20	2
3D	MUL		11	1
3E	*			
3F	SWI	Implicite	19	1
40	NEGA	Implicite	2	1
41	*			
42	*			
43	COMA		2	1
44	LSRA		2	1
45	*			
46	RORA		2	1
47	ASRA		2	1
48	ASLA LSLA		2	1
49	ROLA		2	1
4A	DECA		2	1
4B	*			
4C	INCA		2	1
4D	TSTA		2	1
4E	*			
4F	CLRA	Implicite	2	1
50	NEGB	Implicite	2	1
51	*			
52	*			
53	COMB		2	1
54	LSRB		2	1
55	*			
56	*			
57	RORB		2	1
58	ASRA		2	1
59	ASLB LSLB		2	1
5A	ROLB		2	1
5B	DECB		2	1
5C	*			
5D	INCB		2	1
5E	TSTB		2	1
5F	*			
5F	CLRB	Implicite	2	1

Code OP	Mnémonique	Mode	~	#
60	NEG	Indexé	6•	2•
61	*			
62	*			
63	COM		6•	2•
64	LSR		6•	2•
65	*			
66	ROR		6•	2•
67	ASR		6•	2•
68	ASL/LSL		6•	2•
69	ROL		6•	2•
6A	DEC		6•	2•
6B	*			
6C	INC		6•	2•
6D	TST		6•	2•
6E	JMP		3•	2•
6F	CLR	Indexé	6•	2•
70	NEG	Étendu	7	3
71	*			
72	*			
73	COM		7	3
74	LSR		7	3
75	*			
76	ROR		7	3
77	ASR		7	3
78	ASL LSL		7	3
79	ROL		7	3
7A	DEC		7	3
7B	*			
7C	INC		7	3
7D	TST		7	3
7E	JMP		4	3
7F	CLR	Étendu	7	3
80	SUBA	Immédiat	2	2
81	CMPA		2	2
82	SBCA		2	2
83	SUBD		4	3
84	ANDA		2	2
85	BITA		2	2
86	LDA		2	2
87	*			•#
88	EORA		2	2
89	ADCA		2	2
8A	ORA		2	2
8B	ADDA		2	2
8C	CMPX	Immédiat	4	3
8D	BSR	Relatif	7	2
8E	LDX	Immédiat	3	3
8F	*			

Légende
~ nombre de cycles MPU
# nombre d'octets
* code OP non utilisé

Code OP	Mnémonique	Mode	~	#
90	SUBA	Direct	4	2
91	CMPA		4	2
92	SBCA		4	2
93	SUBD		6	2
94	ANDA		4	2
95	BITA		4	2
96	LDA		4	2
97	STA		4	2
98	EORA		4	2
99	ADCA		4	2
9A	ORA		4	2
9B	ADDA		4	2
9C	CMPX		6	2
9D	JSR		7	2
9E	LDX		5	2
9F	STX	Direct	5	2
A0	SUBA	Indexé	4+	2+
A1	CMPA		4+	2+
A2	SBCA		4+	2+
A3	SUBD		6+	2+
A4	ANDA		4+	2+
A5	BITA		4+	2+
A6	LDA		4+	2+
A7	STA		4+	2+
A8	EORA		4+	2+
A9	ADCA		4+	2+
AA	ORA		4+	2+
AB	ADDA		4+	2+
AC	CMPX		6+	2+
AD	JSR		7+	2+
AE	LDX		5+	2+
AF	STX	Indexé	5+	2+
B0	SUBA	Etendu	5	3
B1	CMPA		5	3
B2	SBCA		5	3
B3	SUBD		7	3
B4	ANDA		5	3
B5	BITA		5	3
B6	LDA		5	3
B7	STA		5	3
B8	EORA		5	3
B9	ADCA		5	3
BA	ORA		5	3
BB	ADDA		5	3
BC	CMPX		7	3
BD	JSR		8	3
BE	LDX		6	3
BF	STX	Etendu	6	3
C0	SUBB	Immédiat	2	2
C1	CMPB		2	2
C2	SBCB		2	2
C3	ADDD		4	3
C4	ANDB		2	2
C5	BITB	Immédiat	2	2
C6	LDB	Immédiat	2	2
C7	*			
C8	EORB		2	2
C9	ADCB		2	2
CA	ORB		2	2
CB	ADDB		2	2
CC	LDD		3	3
CD	*			
CE	LDU	Immédiat	3	3
CF	*			
D0	SUBB	Direct	4	2
D1	CMPB		4	2
D2	SBCB		4	2
D3	ADDD		6	2
D4	ANDB		4	2
D5	BITB		4	2
D6	LDB		4	2
D7	STB		4	2
D8	EORB		4	2
D9	ADCB		4	2
DA	ORB		4	2
DB	ADDB		4	2
DC	LDD		5	2
DD	STD		5	2
DE	LDU		5	2
DF	STU	Direct	5	2
E0	SUBB	Indexé	4+	2+
E1	CMPB		4+	2+
E2	SBCB		4+	2+
E3	ADDD		6+	2+
E4	ANDB		4+	2+
E5	BITB		4+	2+
E6	LDB		4+	2+
E7	STB		4+	2+
E8	EORB		4+	2+
E9	ADCB		4+	2+
EA	ORB		4+	2+
EB	ADDB		4+	2+
EC	LDD		5+	2+
ED	STD		5+	2+
EE	LDU		5+	2+
EF	STU	Indexé	5+	2+
F0	SUBB	Etendu	5	3
F1	CMPB		5	3
F2	SBCB		5	3
F3	ADDD		7	3
F4	ANDB		5	3
F5	BITB		5	3
F6	LDB		5	3
F7	STB		5	3
F8	EORB		5	3
F9	ADCB		5	3
FA	ORB		5	3
FB	ADDB	Etendu	5	3
FC	LDD	Etendu	6	3
FD	STD		6	3
FE	LDU		6	3
FF	STU	Indexé	6	3

Code OP	Mnémonique	Mode	~	#
1021	LBRN	Relatif	5	4
1022	LBHI		5(6)	4
1023	LBLS		5(6)	4
1024	LBHS LBCC		5(6)	4
1025	LBCS LBLO		5(6)	4
1026	LBNE		5(6)	4
1027	LBEQ		5(6)	4
1028	LBVC		5(6)	4
1029	LBVS		5(6)	4
102A	LBPL		5(6)	4
102B	LBMI		5(6)	4
102C	LBGE		5(6)	4
102D	LBLT	Relatif	5(6)	4
102E	LBGT	Relatif	5(6)	4
102F	LBLE	Relatif	5(6)	4
103F	SWI 2	Implicite	20	2
1083	CMPD	Immédiat	5	4
108C	CMPY		5	4
108E	LDY	Immédiat	4	4
1093	CMPD	Direct	7	3
109C	CMPY		7	3
109E	LDY		6	3
109F	STY	Direct	6	3
10A3	CMPD	Indexé	7+	3+
10AC	CMPY		7+	3+
10AE	LDY		6+	3+
10AF	STY	Indexé	6+	3+
10B3	CMPD	Etendu	8	4
10BC	CMPY		8	4
10BE	LDY		7	4
10BF	STY	Etendu	7	4
10CE	LDS	Immédiat	4	4
10DE	LDS	Direct	6	3
10DF	STS	Direct	6	3
10EE	LDS	Indexé	6+	3+
10EF	STS	Indexé	6+	3+
10FE	LDS	Etendu	7	4
10FF	STS	Etendu	7	4
113F	SWI 3	Implicite	20	2
1183	CMPU	Immédiat	5	4
118C	CMPS	Immédiat	5	4
1193	CMPU	Direct	7	3
119C	CMPS	Direct	7	3
11A3	CMPU	Indexé	7+	3+
11AC	CMPS	Indexé	7+	3+
11B3	CMPU	Etendu	8	4
11BC	CMPS	Etendu	8	4

NOTE : Tous les codes opération inutilisés sont indéfinis et interdits

Dans la même collection

Initiation au BASIC TO7/TO7-70
Christine et François-Marie Blondel
Le BASIC D.O.S. du TO7/TO7.70 et du MO5
Christine et François-Marie Blondel
Un ordinateur à la maison
Jean Delcourt
Un ordinateur en fête
Serge Pouts-Lajus
Un ordinateur et des jeux
Jean-Pascal Duclos
Guide pratique de l'ordinateur personnel d'IBM
Dalloz/Emery/Portefaix/Boisgontier/Salzman
LOGO, des ailes pour l'esprit
Horacio C. Reggini
Premiers pas avec le ZX-SPECTRUM
Ian Stewart/Robin Jones
Le langage machine de ZX-SPECTRUM
Ian Stewart/Robin Jones
Plus loin avec le ZX-SPECTRUM
Ian Stewart/Robin Jones
Jeux vidéo, jeux de demain
Georges-Marie Becherraz/Alain Graber
Guide pratique de l'Oric-Atmos
Michel Bussac/Robert Lagoutte
Des programmes pour votre Oric-Atmos
Michel Piot
Premiers pas avec le Commodore 64
Ian Stewart/Robin Jones
Initiation à LOGO
Doris Avram/Michèle Weidenfeld
LOGO, Manuel de référence
Doris Avram/Tristan Savatier/Michèle Weidenfeld et l'équipe de S.O.L.I
Guide du MO5
André Deledicq
Faites vos jeux en assembleur sur TO7 et TO7-70
Michel Oury
Manuel de l'Assembleur du 6809 et du TO7 et TO7-70
Michel Weissgerber
Initiation au FORTH
S.E.F.I.
Guide pratique du vidéotex et du minitel
Jean-Pierre Sabouréau/Geneviève Bouché
Guide pratique de l'enseignement assisté par ordinateur
Jean-Michel Lefèvre

IMPRIMERIE AUBIN, 86240 LIGUGÉ
D.L., novembre 1984. — Impr., L 17363
Imprimé en France